This mock proposal is just an example for current DFG template valid from October 2011.

Neuantrag auf Sachbeihilfe

???Proposal Title???

Acronym: iPoWr

April 30, 2017

Michael Kohlhase, Jacobs University
Great Communicator, Power Consulting

Contents

1 Management

hi to you!

2 Document Management System

2.1 Requirements

The Contractor shall provide a Document Management System which includes the ability to house electronically-filed court documents and auto-relate them to a case index, the ability to scan images and auto-relate them to a case index, the ability to apply digital annotations to imaged documents as part of agile workflow, the ability to auto-post Orders created from the Judicial Dashboard into the case index in real-time, and to keyword search and redact elements inside digital images housed in the Document Management System.

2.2 Breakdown

- Storage of electronic court documents
- indexing of court documents to a case, presumably with a pre-existing id
- ability to scan images
- automatically relate scans to case index
- apply digital annotations to imaged documents
- application of digital annotations must be part of agile[1]
- push orders from Judicial Dashboard into the case index
- keyword search
- redact elements of the scans

2.3 Necessarily Implied By Requirements

The requirement here seems to be an indexing system that needs to plug into an existing id. This sounds like an SQL DB tracking files in some kind of data store. Presumably there needs to be a way to track and apply edits, which will imply some kind of frontend for interpreting edits.

Redactions There also appears to be a request for some kind of editor to "apply redactions." Generally, if redactions are required, we should expect this to be firm and probably important. This has security implications. How do we do a redaction that truly honors the requirement that the information so redacted is actually inaccessible? It seems to me that a document server that serves this will need to store an original and apply redactions **before** the file leaves the server. This further implies that the system will need some kind of authorization and some kind of access level tracking. It will need to be verifiably secure, as well.

Annotations The requirement that digital annotations be applied to an imaged document seems like a common-ish concept. This clearly requires both frontend support and backend support. You will need a method for applying annotations, and a method for capturing these annotations and associating them with a particular digital document. Typical annotations systems will tend to offer a text box, which really boils down to a top-left position for the start of the box, and a width. The rest of the box would be implied by the amount of text written. This would probably constitute a simple version, and later versions (if necessary) could then perhaps talk about such things as font and point size. This is only if we implement this ourselves, although given the specific nature of system it seems like the sort of thing that would need to be custom built.

[1] unclear what is meant by this

Relation to Case Index A number of these requirements allude to association with a case index. Presumeably this already exists, since it sounds like something that the court is using to track cases that exist outside the system and not mearly some nicety of an existing system. However, this charge is unclear, since it could mean anything from enter a case id into the system to get an AI system to figure out what case ID is appropriate to "gee you have this case id open, why don't I associate this document you're adding to this case id?" Of the three possibilities, I consider the last possibility to be the most likely.

In any case, it means at the very least that redactions, annotations, and images must all refer to a case index, and that seems to be the central idea they are trying to get across.

Keyword Search The way this is phrased it sounds like what they want is a way to keyword search scanned documents. This would need to be an OCR system. Otherwise, if they are just trying to keyword search metadata and annotations, this would be trivial. Maybe this is indeed what they want? In any case, there are some IR packages out there that probably aren't too bad. I mean how, can you really do worse than MS Outlook?

Document Management System It's not entirely clear what this is. Is this meant to be a service? Or an application? Or both? My guess would be both.

2.4 Solution Overview

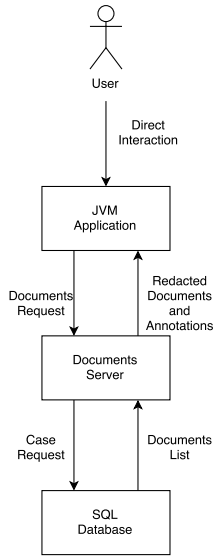

Figure 1: High Level DMS Architecture

An overall solution would involve three main components:

- a client application running on a user's laptop, desktop or other device in a JVM
- an sql document tracker running on a unix-compliant server.
- a document server running on a unix-compliant server.

Client Application The client application would be the main view into the system. This would offer an interface for secure authentication, to be validated by the SQL server and probably some layer on top of it[2].

[2] what existing authentication solutions exist?

The function of this application would be to display images retrieved by the rest of the system, and to apply annotations correctly within the application to the user.

SQL Document Tracker The document tracker would resolve which documents are associated with which cases, and what access levels are appropriate for each authenticated user given their enrollment in the system. Once this document list is resolved, it can be sent to the document server, which in turn will actually retrieve and transmit the documents to the client application.

Document Management System This subsystem will actually control the documents, their annotations, and redaction deltas. Given a list of documents, it would apply redactions to images directly prior to sending, then pair the resulting image with any annotations. This process would be repeated for all documents in the list, bundled together, and sent back to the requesting client.

On second thought Probably the jvm app would want to talk to the DMS directly, and the DMS would talk to the SQL server. It doesn't make any sense for the DMS to trust the client app's word that the list it got was appropriate for the authenticated user's access level.

Communications It is in general possible to accomplish all said communication with json-based rest calls. For image data, byte streams are allowed, although we would probably want to do something like base-64 encoding to return the image data.

2.5 DMS Architecture

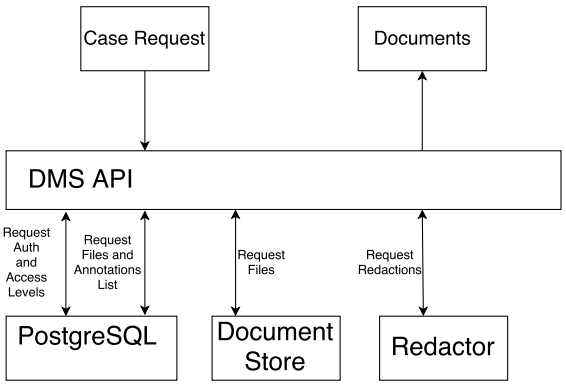

Figure 2: DMS API Architecture

Document Read The DMS API would be divided into several components. There would be an SQL server that tracks image, redaction, and annotation files, and remembers which access levels are associated with those files. It would respond to requests from the DMI API with document lists corresponding to the authentication level of a particular user.

The API would then take that file list and use the descriptors in it to retrieve the appropriate files. These will then be stored in internal memory, and a module, the redactor, would apply redactions to the images. The API would then bundle the redacted image with the annotations, and reply to the client's initial request.

Document Write The DMS API in the case of a write request would verify authentication. If authentication passes, any new documents along with redaction deltas would be stored at the requestor's access level for later processing, and appropriate entry would be made in the database.

Document Update Since this is an authenticated system, and no need for storage of document history was specified, an update would simply be a write with new state. However, since there is a redactor service in place, this would be a convenient module in which to apply deltas and so restore a specific state, if document history retrieval is a desired feature.

2.6 Feasibility

Overall, this architecture is emminently implementable with any of the jvm languages. Jvm languages would be suitable for both client application and API implementation. Off-the-shelf SQL servers, such as PostgreSQL can handle substantially more traffic than a courthouse is likely to receive, and such database technologies are industry standard with robust track records of performance and reliability, as well as many experts able to develop for them and service them.

In general, the primary point of concern is an authentication system, which is under-specified here. It is likely that communicating over TLS and using techniques such as nonces would provide security adequate for the confidential nature of the information in this system, however a detailed review of this is advised. Nevertheless, there is nothing in the requirements that falls outside the applicability of tried and tested security techniques.

2.7 Resources

Resources required for the implementation of this system are likely to be itemized as follows:

Expertise

- Client application: 1 - 2 mid to senior level programmers skilled with JVM languages and frontend development.
- DMS API: 1 - 2 mid to senior level programmers skilled with JVM languages and API development.
- PostgreSQL: 1 mid to senior level DBA with experience in PostgreSQL.
- Security: 1 senior level developer with a focus in security.
- Coordination and leadership: 1 senior level JVM language expert with experience in image processing. This should suffice for coordination and implementation of the redactor module.

Timeframe With the crew listed above, I would expect this subsystem to be implementable within 3-6 months.

3 Electronic Filing Portal and Electronic Filing Fee Payment System

3.1 notes

Pre-existing methods of accepting payments online:

- PayPal offers an integration for websites
- Paysimple is a package that I know little about: https://paysimple.com/
- https://www.2checkout.com/
- there are certainly others

My intuition is that PayPal is going to be the most robust online payments system. There's an API that I don't fully understand, but assuming we can get a token from it that says that a payment has been processed, then we can associate that with the payment that was made, put it in the db, and do what we like with it. So in one sense, we can view the database as the interface with the rest of the system. Probably we want to keep the payments db isolated from the rest of the system.

There are api sandboxes for testing api integration with the system.

From our perspective: there are multiple oppotunities for terrible liabilities if we implement our own financial system. Definitely outsource this.

Stated requirements "The Contractor shall provide a Financial Management System which includes robust accounting functionality, to include: an electronic fee payment system that is fully-integrated with the general ledger, comprehensive transactional audit trail functionality, agile calculation structures, full receipting, sophisticated debits and credits reporting functionality, including established state-financial reporting structures, whereby all payments and disbursements are auto-related into the case ledger, in real time."

Thus, there is:

- Financial transaction processing
- A general ledger
- Audit log
- Calculation system (assuming this means accounting software)
- Receipting (this is most certainly handled by paypal)
- State-financial accounting practices unification
- Rules for relating payments and disbursements in the ledger

3.2 Rough design

So it appears that there's 3 basic layers to this.

There's an interface with whatever payments system we end up using. Whatever that payment system doesn't really matter. The idea is to make sure there is some system that is unified with a database that is a canonical record of transactions. The mission of this system should be to answer the question "Was this payment processed in this amount at this time" with perfect correctness with respect to actual money that changed hands. This system should be a ledger system only. It should expose a read-only authenticated API for reading transaction data. This transaction data should then be replicated to a secondary system. This replication should be secure and one-way. Once replicated at the target system, this target system would consitute the basis for the financial application component of the overall delivered system.

This second layer would then contain all the services for calculation, display of ledger, unification of state financial accounting practices, rules for relating payments and suchlike. The mission of this component should be to unify the transaction data from primary to the model that is expected for end-user interaction where the end-user is the accountant or other financial administrator or authorized employee.

3.3 Design elaboration

Referring to figure 3, we see a number of components.

Buyer / **Fee Payer** this denotes the consumer of the system that will pay fees to the client. This layer need not be a direct interaction with the API, but may encompass a web page that interacts with the API, or a desktop application on a dedicated terminal, or even a mobile application.

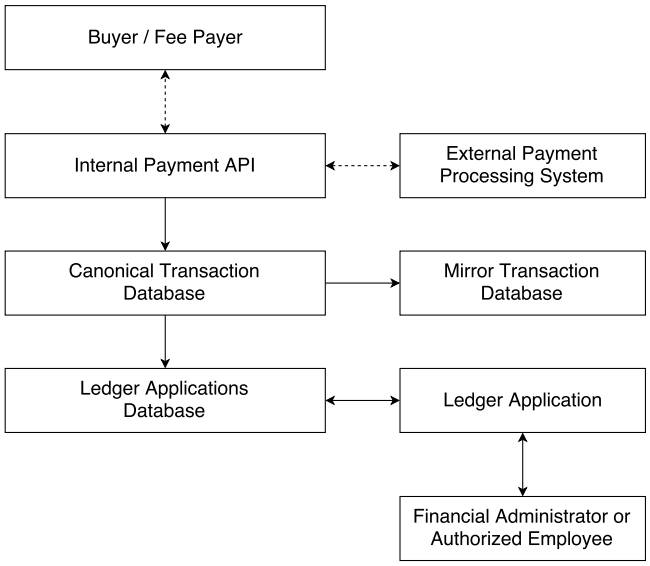

Figure 3: High Level DMS Financial System Architecture

Internal Payment API *this is probably badly named.* Even so, this is the API that processes intention to pay a fee. It is held to be in communication with some external payment processing system. It also communicates directly with the canonical transaction database. Its job is to dispatch intention to pay a fee to the external payment processing system, which in turn will reply with evidence that a payment has been processed. If this payment has been processed, then the canonical transaction database is notified.

Canonical Transaction Database This is considered to be the master authority on whether or not a transaction has taken place. It is to store **only** transaction records. Once written, no record is **ever** to be removed. Any updates, fixups, or retractions must be entered as a separate transaction. Rows must all be double entry, indicating a quantity removed from an account for every quantity deposited to an account, and these quantities must always match. If there is a mistake recorded in this database, it is to **remain** and be corrected with an updating transaction. It will respond to no requests for any of its data, except for the two databases that are to replicate it.

Mirror Transaction Database todo: write me!

Ledger Application Database todo: write me!

Ledger Application todo: write me!

Financial Administrator or Authorized Employee todo: write me! It is often good to separate the top-level sections into separate files. Especially in collaborative proposals. We do this here.

4 Bibliography concerning the state of the art, the research objectives, and the work programme

ToDo:1

In this bibliography, list only the works you cite in your presentation of the state of the art, the research objectives, and the work programme. This bibliography is not the list of publications. Non-published works must be included with the proposal.

Done:1

[1] TO DO: *from the proposal template*