Database design v2-0

work in progress

New database design for version 2.0 - specifications

Guiding principle:

- a database design that makes it easy to support multiple databases. That means no complex queries and no engine specific data types. The application is so simple that complex queries should not be needed anyway.
- a database design that easily explains what data we gather and why
- a database design that is easy to use with the intended class design and makes it easy to expand on in the years to come
- a database design which make update as easy and scriptable as possible

Up until now (version 1.6 release, version 2.0-alpha-summer2013) the database design was effectively based on 2 tables: one for logs and one for everything else. This has served us well but also has some issues. It's hard for new coders to figure out what data is used for what purpose. It is relatively hard to come up with ways to incorporate new functionality in a very restricted database environment. And because everything uses the same record structure the chances that new functionality in for example guest use vouchers introduces issues in the basic file transfer functionality are high. With every new feature there is a significant chance of collateral damage.

The new database design intends to fix these issues. It also intends to make some functionality clearer than it was before. A good example of this is the new auditlog table. When FileSender launched its first prototype we did not know the audit trail capability was a feature that was of crucial importance to most users. It turns out our user base really cares what happens with their files. It seems like a good idea to redesign the audit trail data model and ensure we store all we need to provide a solid trail to the end user.

Tables

Tables related to core functionality: transferring files. The basic component is a transfer. A transfer can have one or more files being transferred. A transfer can have one or more recipients. All activity regarding a transfer, its files or recipients will be logged in the auditlog and statlog tables. Note that the transfer table is proposed to have an "options" field with key-value pairs. That way we can create new options without changing the database.

- **Transfers:** the container for files, recipients and transfer-related meta-data. Transfer owner (UID from IdP), sender's name, message, subject, expiry data (introduce a valid from and a valid until? Useful for public procurement processes :), status (in upload, available for download, expired, etc. figure out what works), options. The options field will contain key-value pairs (JSON format?) for all options like which emails to send, fileId, recipientIds, etc.
- **Files**: file name, file size, file integrity check code (?), transaction Id, field, ?
- **Recipients:** stores all information about recipients. Unique recipient Id, Email address, download URL/token, whether or not the "send email when download complete" option is available for this user, date when the recipient was added, whether or not the recipient is active

I included the auditlog as a table related to core functionality: any transaction on a file can not be considered completed until the audit log has been written (!). This is a direct consequence of wanting to be able to offer a solid guarantee to users the audit log provides an accurate overview of everything that happened with a file.

- **Auditlogs:** stores the audit trail information. Everything that happens to/with a file, voucher or user record is stored here. Events that are related to -
  - a single file (upload, download, removal)
  - a group of files (i.e. a transfer)
  - a user's activity (has the user not made transfers in a long time? Is it time to remove their data?)
  - a voucher (has the recipient used it? Is it expired, canceled then removed?)
  - When a USER is removed, then so is the "catalog" of his recipients (their data in the DB). Not practical to let recipient records expire while user is still active.

Supporting tables:

- **Users:** this is **NOT** a user database. This table stores user preferences and defaults to allow us to tick the option boxes the user wants ticket upon log on. User UID (as received from IdP), which options a user usually has set. Must include a field on whether or not the AuP tick was set and when that was done. Has an implication on data handling agreements etc. Ideally the AuP is ticket on first-logon and then the AuP

question only comes up again when the AuP is changed.
- **Guestvouchers:** stores all information related to the guest usage authentication voucher functionality. Owner UID, recipient, validity period, number of transactions the voucher is good for, optional message, subject, the actual login token, whether or not all options are accessible for the voucher user, etc. The table's functionality could be split but that likely leads to more complexity than is worth the bother.

- **Statslogs:** stores an anonymised version of the audit trail information which can be used for generating statistics

About the life time of tuples in the database

- Entries in the transfers, files and vouchers tables exist for only as long as the actual transfer or auditlog exists: once a transfer and its auditlogs are expired they should disappear from the database table.
- Entries in the user profile should to be subject to a delete/deprovisioning policy
- Entries in the audit log should not store the message and subject, and can in principle be removed once the file transaction expires or the voucher is no longer valid. There should be a policy indicating when entries are purged (ties in with option to send user audit log or not, and specify an audit-trail-retention-time in the config). A good new feature would be to allow a user to receive the audit trail for its transaction(s) when a transaction expires. That way we can send the user a (signed?) PDF (?) and after that delete the audit trail on the server thus reducing the privacy footprint.
- Entries in the statistics log can in principle live on forever. A purge policy should be possible to specify (rentention time = x, 0 for forever? What with no retention time?)

Exact table structure

Tables directly needed for file transfer

table: Transfers

Field name	Data type	Description
id	integer, unsigned	Primary key. Not null and auto-incrementing
trackingid	string(10)	Tracking code for the transfer (sent to sender) **Maybe not that useful**
user_id	string(255)	The sender's user ID, as generated by the IdP
subject	string(255)	Optional: specifies subject in the e-mail notification sent to receivers
message	text	Optional: message from sender to receivers
created	datetime	The date and time of the transaction
expires	datetime	The expiration date and time (to insure that download is available on the last day)
status	string(32)	Indicates status of the file transfer: uploading, available for download, expired, closed
options	text	JSON-encoded that contains data about what options were checked

table: Files

Field name	Data type	Description
id	integer, unsigned	Primary key. Not null and auto-incremented.
transfer_id	integer, unsigned	Foreign key. Not null. The ID handle of the transaction record the file is associated with.
name	string(255)	Name of the file, as originally specified by sender.
size	big integer, unsigned	File size in bytes.
sha1	string(40)	Hash code (for file integrity verification). Should be done per chunk. But leaving it as it is for now.

table: Recipients

Field name	Data type	Description
id	integer, unsigned	Primary key. Not null and auto-incrementing
transfer_id	integer, unsigned	Foreign key. Not null. ID handle for the transfer the recipient is in.
email	string(255)	The recipient's e-mail.
token	string(60)	Download or voucher token for each recipient. Only the latest token for a recipient is shown here.
options	text	JSON encoded value that tells whether recipient has asked to get "download complete" e-mail or other options
created	datetime	The date when the recipient record was created

last_activity	datetime	Date of last activity. Used to determine if recipient is inactive and data should be removed

table: Auditlogs

Field name	Data type	Description
id	integer, unsigned	Primary key. Not null and auto-incrementing
event	string(32)	The type of event that is logged: UPLOAD, FAILED, DOWNLOAD, UPLOADED, LOG_CREATED USER_ACTIVATED, USER_INACTIVE, USER_PURGED FILE_UPDATED, FILE_EXPIRED, FILE_MOVED, FILE_DELETED GUESTVOUCHER_CREATED, GUESTVOUCHER_SENT, GUESTVOUCHER_USED, GUESTVOUCHER_EXPIRED, GUESTVOUCHER_CANCEL, GUESTVOUCHER_CLOSED TRANSFER_START, TRANSFER_END, TRANSFER_CLOSED, TRANSFER_DELETED UPLOAD_START, UPLOAD_END DOWNLOAD_START, DOWNLOAD_END, DOWNLOAD_RESUME
user_id	string(255)	Id of the user who triggered the event
target_id	integer, unsigned	The target of the event
target_type	string(255)	Type of the target of the event : Transfer, File, Guestvoucher ...
created	datetime	The date and time the event occured
ip	string(36)	The IP of the user (be it v4 or v6)

Supporting tables

table: Guestvouchers

Field name	Data type	Description
id	integer, unsigned	Primary key. Not null and auto-incrementing
user_id	string(255)	The user who has issued the voucher/under whose name the transfer will be done.
token	string(60)	The voucher token
transfers	integer, unsigned	Nr of transfers the voucher is good for.
created	datetime	Date the voucher token was created and sent
expires	datetime	Specifies the expiration date and time of the voucher
subject	string(255)	The subject of the "invitation by voucher" e-mail message
message	text	The message body of an "invitation by voucher"
options	text	JSON encoded options which are accessible to voucher users when sending files (Initially ignored)

table: Users

Field name	**Data type**	**Description**
id	string(255)	From IdP authorization step, the user's ID
organization	string(80)	Organization the user is associated with. From IdP authorization step, can be null
aup_ticked	boolean	Whether the user's already accepted the AuP conditions by ticking the box
aup_last_ticked_date	datetime	date the AuP was last ticked by the user. Allows us to figure out if this user needs to tick the AuP again due to an AuP update. Such an update should reset all users' "aup_ticked" property.
transfer_preferences	text	text string, JSON-encoded, that contains data about what options user had previously set
voucher_preferences	text	Not used currently, since there are no options to select.
created	datetime	Date and time of user preference record creation
last_activity	datetime	Date and time of last user activity

table: Statslogs - anonymized version of Auditlogs. E.g. fields such as file name replaced by, for instance, file ID (integer), which will be meaningless once the transfer expires and the related file records are removed - then fileID is indeed anonymous.

Field name	Data type	Description
id	integer,	Primary key, not null, auto-increment

id	unsigned	Primary key, not null, auto increment
event	string(32)	The type of event that is logged: UPLOAD, FAILED, DOWNLOAD, UPLOADED, LOG_CREATED USER_ACTIVATED, USER_INACTIVE, USER_PURGED FILE_UPDATED, FILE_EXPIRED, FILE_MOVED, FILE_DELETED GUESTVOUCHER_CREATED, GUESTVOUCHER_SENT, GUESTVOUCHER_USED, GUESTVOUCHER_EXPIRED, GUESTVOUCHER_CANCEL, GUESTVOUCHER_CLOSED TRANSFER_START, TRANSFER_END, TRANSFER_CLOSED, TRANSFER_DELETED UPLOAD_START, UPLOAD_END DOWNLOAD_START, DOWNLOAD_END, DOWNLOAD_RESUME
target_type	string(32)	Type of the target of the event : Transfer, File, Guestvoucher ...
size	big integer, unsigned	Size of event target
created	datetime	Date and time the event occured

For reference: the database design inherited from Filesender v2.0 alpha (and most of it from v1.x)

file table:

FIELD NAME	DATA TYPE	DESCRIPTION
fileto	text	Receiver's email address.
filesubject	character varying(250)	Title, used as part of subject in emails.
filevoucheruid	character varying(60)	Unique file id based on the combination of "file" and "to". This is used for download links.
filemessage	text	Message from the sender to the receiver, used as part of body in emails.
filefrom	character varying(250)	Sender's email address.
filesize	bigint	The size of the file, in bytes.
fileoriginalname	character varying(500)	The name of the file.
filestatus	character varying(60)	The status of the file, can be any one of: "", "Voucher", "Voucher Cancelled", "Closed", "Deleted".
fileip4address	character varying(24)	Sender's IPv4 address.
fileip6address	character varying(45)	Sender's IPv6 address.
filesendersname	character varying(250)	Never used?
filereceiversname	character varying(250)	Never used?
filevouchertype	character varying(60)	Never used?
fileuid	character varying(60)	Unique file id.
fileid	**integer**	**Primary key. Auto incrementing number.**
fileexpirydate	timestamp without time zone	The date and time at which the file will expire.

fileactivitydate	timestamp without time zone	The date and time at which the file entry was last updated.
fileauthuseruid	character varying(500)	Uniquely identifies users based on their authentication.
filecreateddate	timestamp without time zone	The date and time at which the file was uploaded.
fileauthurl	character varying(500)	Never used?
fileauthuseremail	character varying(255)	User's email address, based on their authentication data.
filegroupid	character varying (60)	a multi-file transaction consists of individual files (and file records) with the same groupid
filetrackingcode	character varying (5)	used as a user-friendly way to identify a multi-file transaction
filedownloadconfirmations	character varying (5)	unsure
fileenabledownloadreceipts	character varying (5)	probably used to let a sender indicate a receiver is allowed to indicate they want to receive an email receipt once the download is complete
filedailysummary	character varying (5)	used to indicated whether or not the sender of a file wants to have a daily summery of his filesender activity (uploads, downloads)
filenumdownloads	integer default 0	
downloaded_by_sender		? to indicate the sender downloaded a (group of) files directly from MyFiles

log

COLUMN NAME	DATA TYPE	DESCRIPTION
logid	**integer**	**Primary key. Auto incrementing number.**
logfileuid	character varying(60)	Unique file id.
logtype	character varying(60)	The type of logged event. Can be any one of the following: Voucher Cancelled, Uploaded, File Removed (Expired), Error, File Moved, Updated, Upload Attempt, Download, Voucher Sent, File Cancelled, Voucher Closed.
logfrom	character varying(250)	Sender's email address.
logto	text	Receiver's email address.
logdate	timestamp without time zone	The date and time at which this event was logged.
logtime	time with time zone	Deprecated field, no longer used.
logfilesize	bigint	The size of the file connected to this log entry, in bytes.
logfilename	character varying(250)	The name of the file connected to this log entry.
logsessionid	character varying(60)	Never used?
logmessage	text	A message describing the reason for the log entry.
logvoucheruid	character varying(60)	Unique file id based on the combination of "file" and "to". This is used for download links.
logauthuseruid	character varying(500)	Uniquely identifies users based on their authentication.
logfilegroupid	character varying (60)	
logfiletrackingcode	character varying (5)	
logdailysummary	character varying (5)	