Project team # - Oauth Database Project

Team Members: Tommy, Lansina

## PROJECT PROPOSAL

**Content, Scope and Objectives**

OAuth systems must manage information about tokens handed to the client by the provider (ex: Google). Such as access/refresh token, client ID/Secret, Token binding (roles), and expiration times.

## PROJECT ENVIRONMENT

Authentication provides system database for managing client and session information. There is a master table which holds information about the system owner. There is another table designed for the system owner to distribute their own keys with certain attributes. This allows for the owner to distribute authentication without ever giving admin access to anyone but themselves. This also allows the OAuth system managers to have a simple 2 table setup for many admins and users with each client being associated with a Master Entity and having their own special access token.

## HIGH LEVEL REQUIREMENTS

### Initial user roles

|  |  |
| --- | --- |
| **User Role** | **Description** |
| Session/Master | Any system requesting token to generate user tokens from |
| Master/Session Users | Stores access tokens, roles, and expiration from the master/session token |

### Initial user story descriptions

|  |  |
| --- | --- |
| **Story ID** | **Story description** |
| Session/Master Developer | I want to allow users to login once with username and password and provide them with a valid token. This token can be used to login on later requests and expires at a certain time past the issue date. This allows users to not have to re-login on each request and allows the system developer to distribute different types of tokens to their users. |
| Master/Session Users | Logging into a website once and then being able to come back and remember that I’m logged in is amazing. My friend also uses a special role that allowed them to do certain things which were awesome to see in action! |

## HIGH LEVEL CONCEPTUAL DESIGN

Entities:

Session/Master:

ID (primary key for database)

Entity ID (name of the owner of the master token)

Master Token (Used to validate other users within that system)

Session/Master Users:

Client ID (generated from the master token is the same as a username)

Access Token/Secret

(Generated from the system when instantiating a new client to authenticate on their application/system)

Role (Role of the users, roles are determined by system manager)

Expires In

(How long a token is valid for before a user is required to relog in again)

Relationships:

Session/Master:

Master Token references Entity ID (one to one)

Session/Master Users:

Client ID references Master Token (many to one)

# Sprint 1

## REQUIREMENTS

Refine the user stories that you made in previous sprint. List your updated user stories and any notes you wish to include in decreasing order of priority and highlight the stories chosen for Sprint 1. *There is no need to show your story refinement process - just the list of updated stories suffices.* Use the format shown below.

|  |  |
| --- | --- |
| **Story ID** | **Story description** |
| US1 | As an admin I want to generator a master token for my entity so I can manage users securely |
| US2 | As an admin I want to assign users with different roles and expiration times, so that users have controlled access. |

## CONCEPTUAL DESIGN

Include your detailed conceptual design here. Use the format shown below.

Entity: **Session/Master**

Attributes:

ID (primary key) - Simple, Single-Valued, Stored

Entity ID – Simple, Single-Valued, Stored, Unique

Master Token – Simple, Single-Valued, Stored, Unique

Entity: **Session/Master Users**

Attributes:

Client ID (Primary Key) - Simple. Single-Valued,Stored

Access Token/Secret - Simple, Single-Valued, Stored

Role - Simple, Single-Valued, Stored

Expiry - Simple, Single-Valued, Stored

CREATED\_AT - Simple, Single-Valued, Stored (default: NOW())

ADMIN\_ID - Simple, Single-Valued, Stored (Foreign Key to ADMIN.ID)

Relationship: **Master/Session** manages **Master/Session Users**

Cardinality: One to Many

Participation:

(A master could have no users but there can be no users without a master)

Entity1 has total participation

Entity2 has partial participation

## LOGICAL DESIGN

Include your logical design here. Use the format shown below.

Table: **Admin**

Columns:

ID - (PK,INT,AUTO\_INCREMENT)

Entity\_ID - (VARCHAR, UNIQUE, NOT NULL)

Master\_Token - (VARCHAR,UNIQUE, NOT NULL)

Table: **Users**

Columns:

Client\_ID - (PK,VARCHAR)

Access\_Token - (VARCHAR,NOT NULL)

Role -(TEXT, NOT NULL)

Expiry - (TIMESTAMP, NOT NULL)

CREATED\_AT - (TIMESTAMP,DEFAULT NOW(), NOT NULL)

ADMIN\_ID -(INT, NOT NULL, FK referencing to ADMIN(ID),ON DELETE CASCADE

The client ID’s will contain information from the master token therefore each client will have an identifier being its respective master token. It must be unique as two clients cannot have the same name as they may be issues the same token which is an issue.

## SQL QUERIES

List at least **three** SQL queries that perform data retrievals relevant to the features chosen in the current sprint. For each query, paste a **screenshot** of the output, as shown through database management tool.

SELECT \* FROM ADMIN, USERS;

SELECT \* FROM ADMIN WHERE ID = 1;

SELECT \* FROM USERS WHERE ADMIN\_ID = 1

Sprint 2

## REQUIREMENTS

Refine the user stories that you made in previous sprint. List your updated user stories in decreasing order of priority. Highlight the stories for which database design was completed in Sprint 1 in one color. Highlight the updated/new stories chosen for Sprint 2 in a different color. *There is no need to explicitly show your story refinement process.* Use the format shown below.

|  |  |
| --- | --- |
| **Story ID** | **Story description** |
| US1 | As an admin I want to remove a user's access token in case I need to remove unauthorized access. |
| US2 | As an admin, I want to view all users under my entity so I can monitor who has access and their roles |
| US3 | As an admin I’d like to check the time of sessions expiring within 24 hours |
| US4 | As an Admin I’d like to be able to extend the time for my moderators session |

## CONCEPTUAL DESIGN

Include your complete updated conceptual design here. Use the format shown below.

Entity: ADMIN

Attributes:

* ID [simple, single-valued, primary key]
* ENTITY\_ID [simple,single-valued,stored,unique]
* MASTER\_TOKEN [simple, single-valued,stored,unique]

Entity: USERS

Attributes:

* CLIENT\_ID [simple, single-valued, stored, primary key]
* ACCESS\_TOKEN [simple, single-valued, stored]
* ROLE [simple,single-valued,stored]
* EXPIRY[simple,single-valued,stored]
* CREATED\_AT[simple,single-valued, stored, default NOW()
* ADMIN\_ID[simple,single-vlaued,stored,foreign key to ADMIN.ID]

Relationship: ADMIN manages USERS

Cardinality: One-to-many, one ADMIN can manage many USERS

Participation:

ADMIN: Total (every USER must be tied to an ADMIN)

USERS: Partial

## LOGICAL DESIGN WITH NORMAL FORM IDENTIFICATION

Include your complete updated logical design here. Use the format shown below.

Table: ADMIN

* ID (PK, INT, AUTO\_INCREMENT, NOT NULL)
* ENTITY\_ID (VARCHAR(255), UNIQUE, NOT NULL)
* MASTER\_TOKEN (VARCHAR(255), UNIQUE, NOT NULL)

Highest normalization level:BCNF

Justification: All attributes are dependent on the primary key

Table: USERS

Columns:

* CLIENT\_ID (PK, VARCHAR(255), NOT NULL)
* ACCESS\_TOKEN (VARCHAR(255), NOT NULL)
* ROLE (TEXT, NOT NULL)
* EXPIRY (TIMESTAMP, NOT NULL)
* CREATED\_AT (TIMESTAMP DEFAULT NOW(), NOT NULL)
* ADMIN\_ID (INT, NOT NULL, FK -> ADMIN.ID, ON DELETE CASCADE)

Highest normalization level: BCNF

Justification All non-key attributes are fully dependent on the PK

## SQL QUERIES

Refine your SQL queries that you designed in the previous sprint if in need. List at least **three** SQL queries that perform data retrievals relevant to the features chosen in the current sprint. For each query, paste a **screenshot** of the output, as shown through your user interface.

Remove token:

DELETE FROM USERS

WHERE CLIENT\_ID = 'User1'; AND ADMIN\_ID = 1;

Check for accounts expiring within 24 hours:

SELECT CLIENT\_ID, ROLE, EXPIRY

FROM USERS

WHERE EXPIRY <= NOW() + INTERVAL 1 DAY;

Extending moderator access time:

UPDATE USERS

SET EXPIRY = NOW() + INTERVAL 7 DAY

WHERE CLIENT\_ID = 'Linus Torvald' AND ROLE = 'MODERATOR';

Sprint 3

## REQUIREMENTS

Refine the user stories that you made in previous sprint. List your updated user stories in decreasing order of priority. Highlight the stories that were completed in Sprint 1 in one color. Highlight the stories that were completed in Sprint 2 in a different color. Highlight the updated/new stories chosen for Sprint 3, if any, in a third color. *There is no need to explicitly show your story refinement process.* Use the format shown below.

|  |  |
| --- | --- |
| **Story ID** | **Story description** |
| US1 | As a system admin, I want a dashboard like view showing all tokens and their statuses so I can manage the system more efficiently. |
| US2 | As an admin, I want to count how many active tokens are still valid passed a given date. |
| US3 | As an admin, I want to automate token cleanup of expired sessions, so that my system remains efficient and secure. |

## CONCEPTUAL DESIGN

Include your complete updated conceptual design here. Use the format shown below.

Entity: **ADMIN**

Attributes:

ID: simple, primary key

ENTITY\_ID: unique name for admin

MASTER\_TOKEN: unique token

Entity: **USERS**

Attributes:

CLIENT\_ID: primary key

ACCESS\_TOKEN: unique token

ROLE: role assigned

EXPIRY: timestamp of token expiration

CREATED\_AT: timestamp of user creation

ADMIN\_ID: foreign key references to ADMIN.ID

Relationship: **ADMIN** manages **USERS**

Cardinality: One-to-Many (one admin to many users)

Participation:

**ADMIN**: Total

**USERS**: Partial

## LOGICAL DESIGN WITH HIGHEST NORMAL FORMS AND INDEXES

Include your complete updated logical design here. Use the format shown below.

Table: **ADMIN**

Columns:

ID INT PK, AUTO\_INCREMENT, NOT NULL

ENTITY\_ID VARCHAR(255) UNIQUE, NOT NULL

MASTER\_TOKEN VARCHAR(255) UNIQUE, NOT NULL

Highest normalization level: BCNF

Indexes:

Index #: Clustered

Columns: ID

Justification: Primary key access for JOINs and foreign key references

Table: **USERS**

Columns:

CLIENT\_ID VARCHAR(255) PK, NOT NULL

ACCESS\_TOKEN VARCHAR(255), NOT NULL

ROLE TEXT, NOT NULL

EXPIRY TIMESTAMP NOT NULL

CREATED\_AT TIMESTAMP DEFAULT NOW(), NOT NULL

ADMIN\_ID INT FK REFERENCES ADMIN.ID, ON DELETE CASCADE

Highest normalization level: BCNF

Indexes:

Index #2: Non-clustered

Columns: EXPIRY

Justification: Used frequently in time-based queries (finding soon to expire tokens)

## VIEWS AND STORED PROGRAMS

List the views relevant to your application here. Use the format specified below.

**View**: TOKEN\_AUDIT\_LOG\_VIEW

Goal: Designed to help admins track token-related activity across the system. It joins data from the AUDIT\_LOG, USERS, and ADMIN table to give a comprehensive summary of actions taken, who was affected, and which admins are responsible.

List the stored programs relevant to your application thus far here. Use the format specified below for the different kinds of stored programs. Note: if you do not have a particular type of stored program in your application, just leave that part out.

**Stored procedure**: remove\_user\_token

Parameters: IN target\_client\_id VARCHAR(255)

Goal: This stored procedure safely removes a user token and logs the deletion in the AUDIT\_LOG. It first finds the related admin, then deleted the user entry and logs this action with a ‘DELETE’ event.

**Stored function**: is\_token\_valid

Parameters: client\_id VARCHAR(255)

Goal: Checks if a users token is still valid (not expired); returns 1 for valid 0 for expired.

**Trigger**: After Insert on USERS

Goal: Automatically log the creation of a new user token to the AUDIT\_LOG table. This is helpful for keeping a history of who granted access and when.

**Event**: daily\_expired\_cleanup

Goal: Removes all expired tokens once per day to maintain system performance and limit unnecessary access.