User Story 1:

@startuml

participant Browser

participant JS

participant Server

participant DB

== SQL evaluation ==

Browser -> Server: GET evaluation-{evalId}(trigger is required Scheduled_atDATETIME)

Server -> DB: GET the details and description of questions from by Select diagram_path from quiz-db

DB -> Server: Returns the diagram details

Server -> Js: 200 if all right

JS -> Browser: Displays start button for evaluation

Server -> DB: GETS question_text-{user_id} by SELECT question_text from sql_question

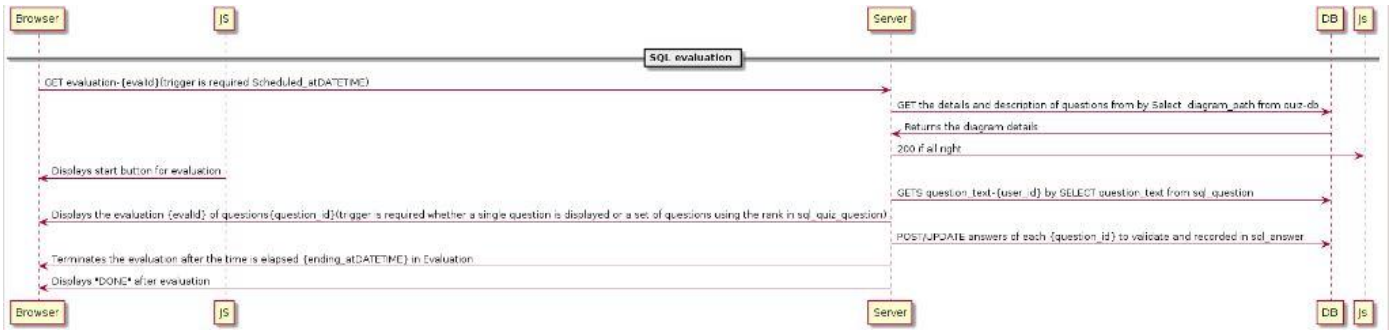
Server -> Browser: Displays the evaluation {evalId} of questions{question_id}(trigger is required whether a single question is displayed or a set of questions using the rank in sql_quiz_question)

Server -> DB: POST/UPDATE answers of each {question_id} to validate and recorded in sql_answer

Server -> Browser: Terminates the evaluation after the time is elapsed {ending_atDATETIME} in Evaluation

Server -> Browser: Displays "DONE" after evaluation

@enduml



Userstory 2:

@startuml

participant Browser

participant JS

participant Server

participant DB

==List of Evaluations==

==verification of trainer==

Browser->Server: GET evaluation-{evalid}{start correcting copies}

Browser->Server: GET sheet-{evaluation_id}{get details from sheet}

Server -> DB: select * from sheet to GET the details of start, end and class of each evaluation (a trigger will be set only if the copies are corrected by the trainer himself by trainer_id)

DB->Server: Return sheet details

Server->Browser: Display sheet details

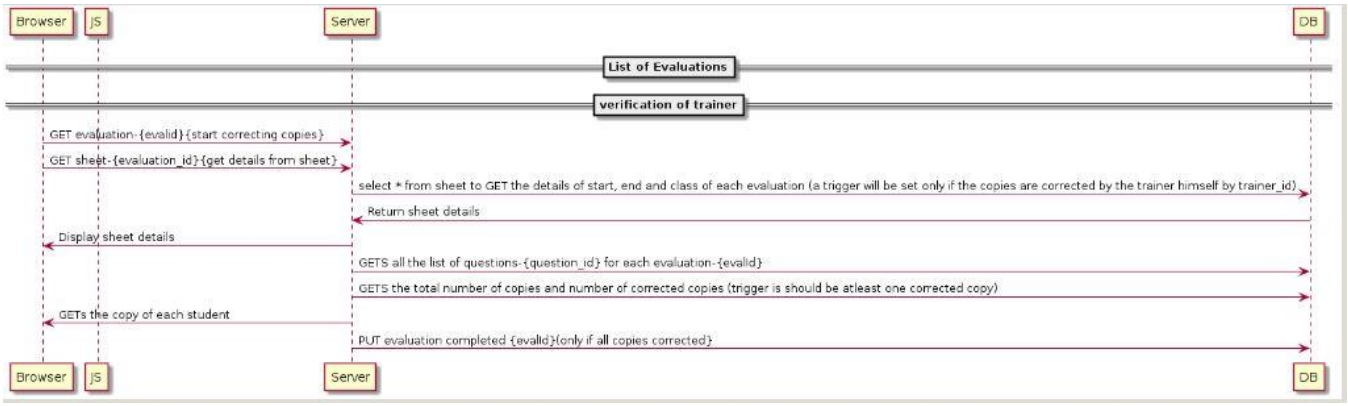
Server -> DB: GETS all the list of questions-{question_id} for each evaluation-{evalId}

Server -> DB: GETS the total number of copies and number of corrected copies (trigger is should be atleast one corrected copy)

Server -> Browser: GETs the copy of each student

Server -> DB: PUT evaluation completed {evalId}(only if all copies corrected}

@enduml



Userstory 3:

@startuml

participant Browser

participant JS

participant Server

participant DB

==online student evaluation==

==retrieve questions ==

Browser -> Server: GET evaluation-{evaluation_id}

Browser-> Server: GET question_text-{trainer_id}, answer_TEXT-{evaluation_id}, correct_answer-{trainer-id}

Server -> DB: SELECT question_text, correct_answer FROM sql_question

Server -> DB: SELECT answer from sql_answer(trigger: at least one validated answer sheet from trainer)

DB->Server: POST question_text-{trainer_id}, answer_TEXT-{evaluation_id}, correct_answer-{trainer-id}

Server->JS: Format into HTML tables, with student query displayed in left and trainer query displayed in right

JS->Browser: POST question_text-{trainer_id}, answer_TEXT-{evaluation_id}, correct_answer-{trainer-id}

Server->JS: 200 if "OK"

JS->Browser: "OK"

==Display validation status by app ==

Browser->Server: GET evaluation-{evaluation_id}

Browser->Server: GET validated_at-{evaluation_id}

Server->DB: SELECT validated_at where this.question_id = question_id

DB->Server: POST validated_at-{evaluation_id}

Server->JS: Return highlighted colors(PALE green/PALE red) around validated_at status

JS->Browser: POST validated_at-{evaluation_id}

Server->JS: 200 if "OK"

JS->Browser: "OK"

==Trainer change validation status ==

Browser->Server: PUT evaluation-{evaluation_id}

Browser->Server: PUT validated_at-{evaluation_id}

Server->DB: UPDATE validated_at where this.question_id = question_id

DB->Server: POST validated_at-{evaluation_id}

Server->JS: Return highlighted colors(green/red) around validated_at status

JS->Browser: POST validated_at-{evaluation_id}

Server->JS: 200 if "OK"

JS->Browser: "OK"

==Display result ==

Browser->Server: GET evaluation--{evaluation_id}

Browser->Server: GET result--{question_id}

Server->DB: SELECT result from training_answer where this.question_id = question_id

DB->Server: POST result--{question-id}

Server->Browser: POST result--{question-id}

Server->JS: 200 if "OK"

JS->Browser: "OK"

==Mark completion of student copy evaluation ==

Browser->Server: POST evaluation--{evaluation_id}

Browser->Server: GET completed_at--{trainer_id}

Server->DB: UPDATE completed_at FROM evaluation where this.trainer_id = trainer_id

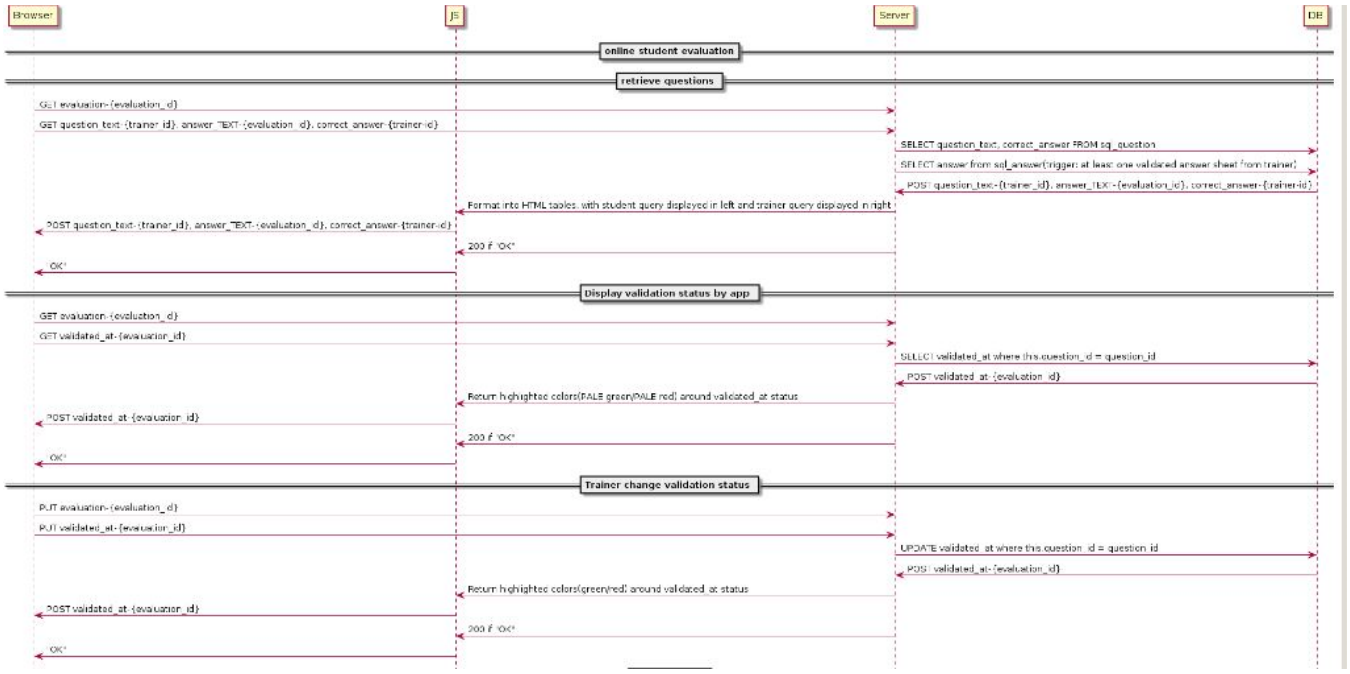
DB->Server: GET completed_at--{trainer_id}

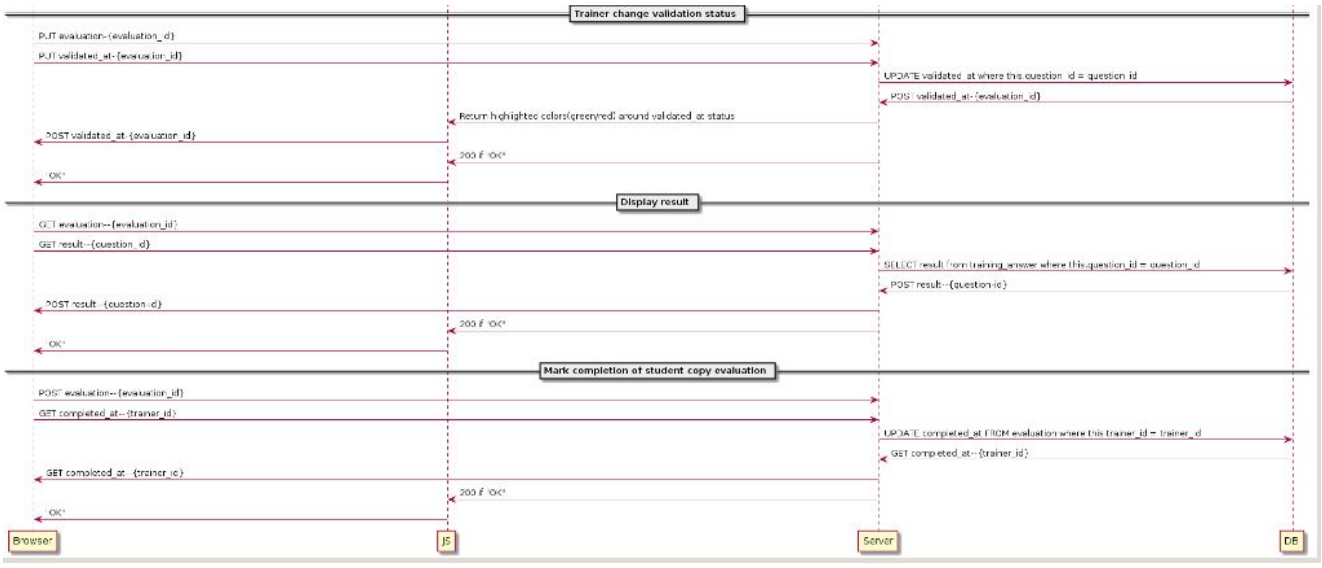
Server->Browser: GET completed_at--{trainer_id}

Server->JS: 200 if "OK"

JS->Browser: "OK"

@enduml





Userstory 4:

@startuml

======Login==

participant Browser

participant JS

participant Server

participant DB

======Correcting online a Question==

Browser -> JS: POST Question data is displayed at the left fixed position

Browser -> Server: Get all questions through question_id {question_id}

Server -> DB: Select question_id, quiz_name from question table

DB -> Server : Fetches the list of questions

Server -> Browser: Displays the list of questions

Browser -> Server: Get the {question text}, result text, answer text, number of correct copies on {question_id}

Server -> DB: Select question_id, question text, result text, answer text from sql_answer

DB -> Server: Fetches the result for the query and displays the result

Server -> Browser: POST the question details and the correct copies

Browser -> JS: POST complete set of answers at right

Server -> DB: Select evaluation_id, result text, gives_correct_result from sql_answer

DB -> Server: fetches the result for the query

Server -> Browser: POST the details student_name, results and status

Server -> DB: Gets the incompleted evaluation_id

Server -> Browser: POST the validated and invalidated Evaluation_id(trigger if eval is incompleted then make it as invalidated)

Server -> DB: PUT/Update the changes immediately after invalidated

Server -> DB: Select * from User

DB -> Server: fetches the User table and returns the result for the query

Server -> Browser: POST details of the User

@enduml

Userstory 5:

@startuml

participant Browser

participant JS

participant Server

participant DB

==student registration==

Browser -> Server:Sends the email and password credentials entered by the student

Server -> DB:If the student is not registered then INSERT/PUT email, password into user;

DB -> Server: creates session variables for email and password

==confirm request==

DB -> Server: Creates email and password for every student

Server -> Browser: POST the confirmation of registration for joining into the class

==class still open==

Server -> DB: checks whether the class is still open to access the evaluations

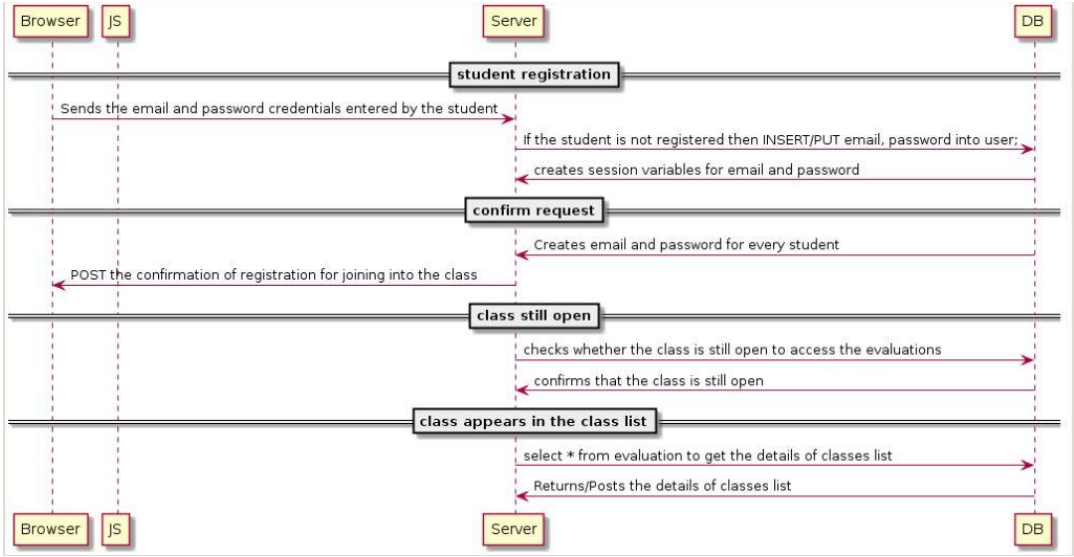
DB -> Server: confirms that the class is still open

==class appears in the class list==

Server -> DB: select * from evaluation to get the details of classes list

DB -> Server: Returns/Posts the details of classes list

@enduml



Userstory 6:

@startuml

== Get list of validated members ==

participant Browser

participant JS

participant Server

participant DB

Browser -> Server: GET list of validated members-{memberid as user_id}

Server -> DB: SELECT {user_id}-{first_name}-{last_name}-{email}-{validated_at} FROM user WHERE ...

DB -> Server: Fetches the list of members

Server -> Browser: Displays list of members

== validated, invalidated and pending requests have an associated style ==

JS -> Server: GET colors at user-{user_id}

Server -> DB: execute SELECT * FROM user WHERE ...

DB -> Server: show all members validated, invalidated, or pending (requires trigger to check status)

Server -> JS: list of members with colors

== validate or invalidate or postpone request to join group ==

Browser -> Server: POST user-{user_id}

Server -> DB: SELECT {user-id}-{validation_at} FROM user WHERE ...

DB -> Browser: Action performed

Server -> Browser: Action has been completed

== Validation recorded ==

JS -> Server: PUT user-{validation_at}

Server -> DB: UPDATE user {user_id}-{validation_at} with system time ...

DB -> Browser: {validation_at} show time

Server -> Browser: Shows the validated time

== close class ==

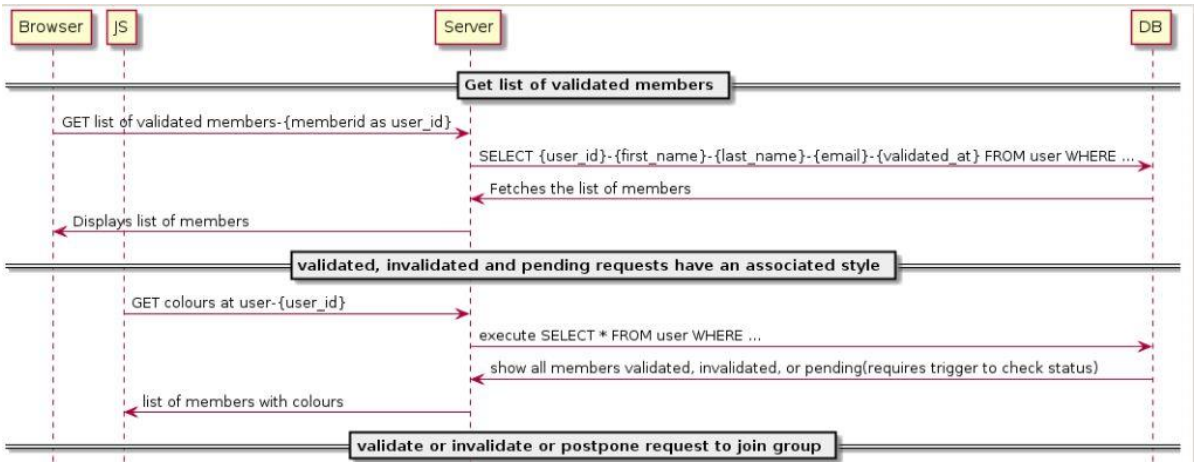
JS -> Server: DELETE user

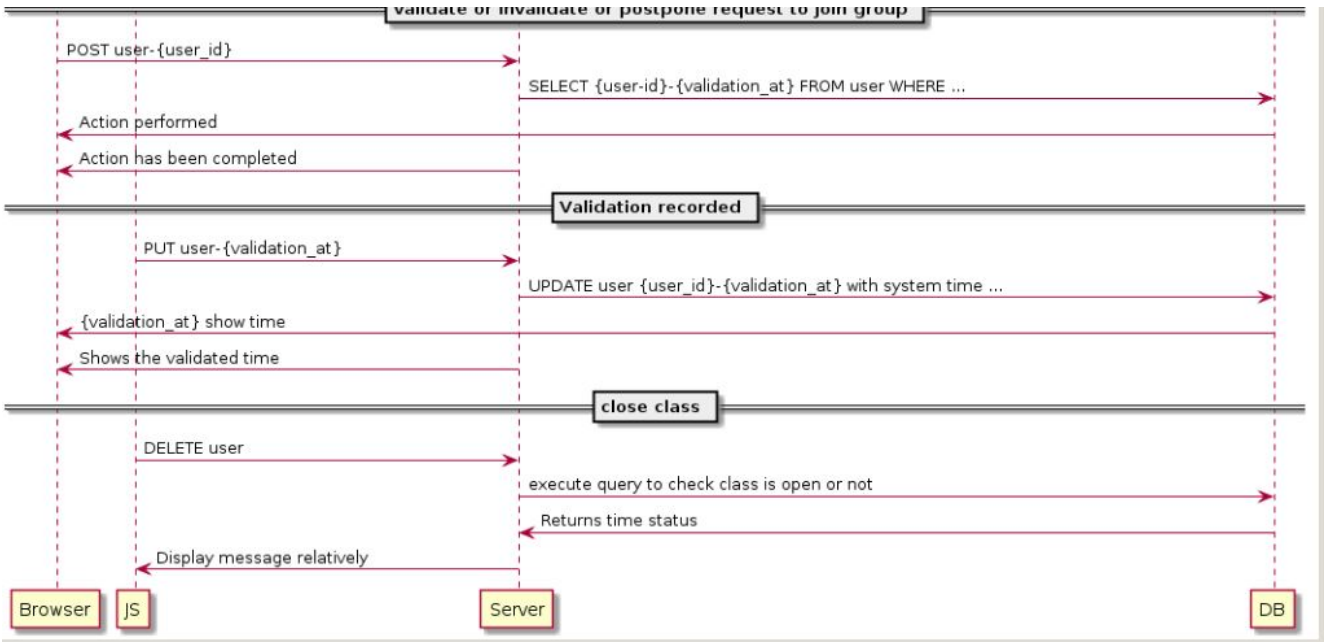
Server -> DB: execute query to check class is open or not

DB -> Server: Returns time status

Server -> JS: Display message relatively

@enduml


Browser
JS
Server
DB
Get list of validated members
GET list of validated members-{memberid as user_id}
SELECT {user_id}-{first_name}-{last_name}-{email}-{validated_at} FROM user WHERE ...
Fetches the list of members
Displays list of members
validated, invalidated and pending requests have an associated style
GET colours at user-{user_id}
execute SELECT * FROM user WHERE ...
show all members validated, invalidated, or pending(requires trigger to check status)
list of members with colours
validate or invalidate or postpone request to join group


validate or invalidate or postpone request to join group
POST user-{user_id}
SELECT {user-id}-{validation_at} FROM user WHERE ...
Action performed
Action has been completed
Validation recorded
PUT user-{validation_at}
UPDATE user {user_id}-{validation_at} with system time ...
{validation_at} show time
Shows the validated time
close class
DELETE user
execute query to check class is open or not
Returns time status
Display message relatively
Browser
JS
Server
DB