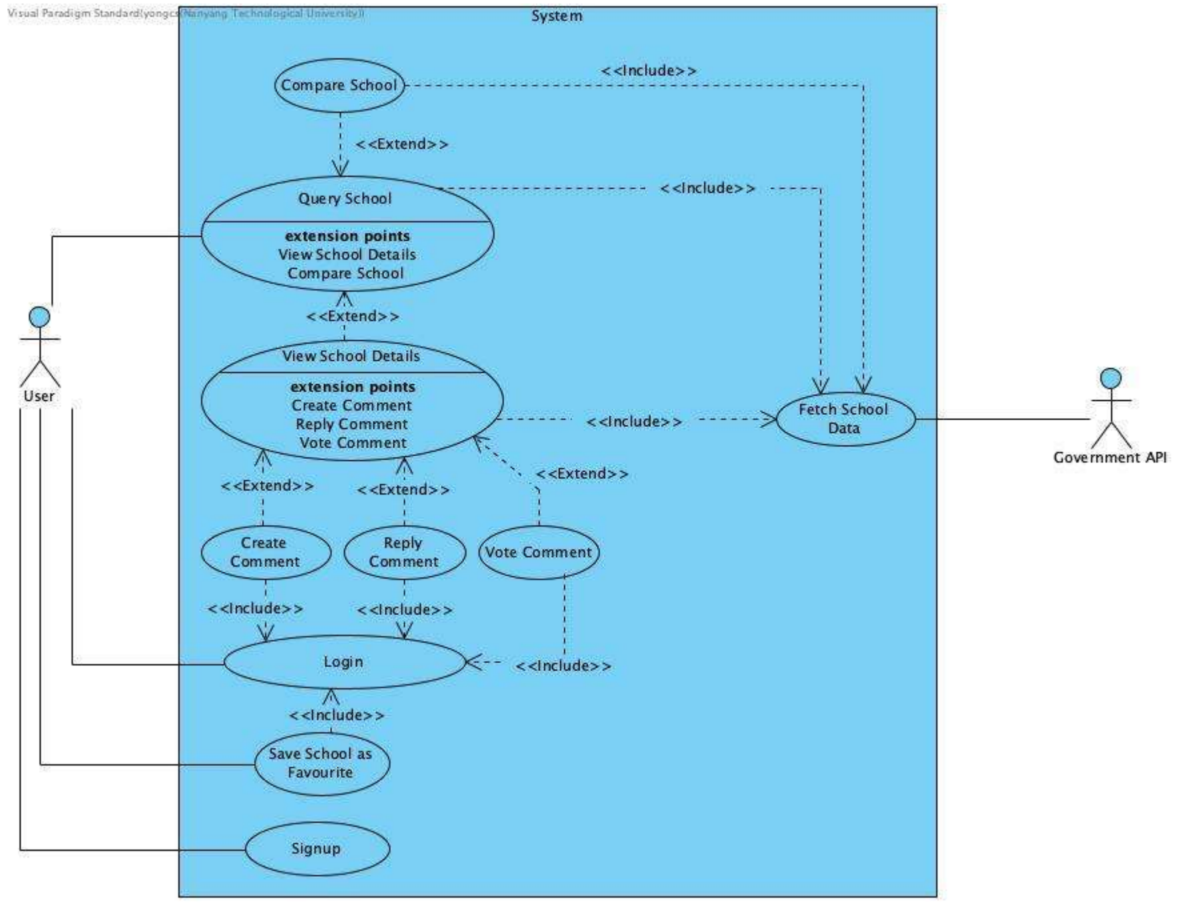
NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

SC2006-Software Engineering

Lab 2 Deliverables

Group Member	**Matric Number**
Lim Kiat Yang Ryan	U2421937D
Yu Wenhao	U2421425F
Yong Chee Seng	U2420563K
Peng Sizhe	U2423895H
Tarun Ilangovan	U2422251A

A. Complete Use Case Diagram



B. Use Case Descriptions

1.1. UCQ

This section describes all use cases related to querying and retrieving a list of results. All use case IDs in this section are prefixed with **UCQ**.

1.1.1 UCQ-1 Query School

Use Case ID:	UCQ-1		
Use Case Name:	Query School		
Created By:	Yong Chee Seng	Last Updated By:	
Date Created:	4 September 2025	Date Last Updated:	

Actor:	User (Initiating)
Description:	User shall be able to query school based on different criteria, and system shall show list of schools that satisfy all the criteria.
Preconditions:	NA
Postconditions:	1. System displays a list of schools that match all selected criteria or shows "no results" if none found.
Priority:	High
Frequency of Use:	Estimated 50–200 per day across all users
Flow of Events:	1. User navigates to "Query School" page. 2. System initiates **UCSY-1 Fetch School Data from Government API** to retrieve the latest school dataset. 3. System displays the full list of schools. 4. System displays available query criteria (e.g., name, region, subject, CCA). 5. User enters one or more criteria. 6. User submits the query. 7. System validates the query input. 8. System filters the school data according to criteria.

	9. System displays a list of matching schools with summary information.
Alternative Flows:	AFS3: **UCSY-1 Throws Exception** 1. System displays error “Unexpected error occurred. Please try again”. AFS8: **No Criteria Entered** 1. System continues showing all schools. AFS9: **No Results Found** 1. If no school satisfies all the criteria, system displays message “No result found”.
Exceptions:	NA
Includes:	1. UCSY-1 Fetch School Data from Government API
Special Requirements:	1. System shall allow selection of criteria from predefined options where applicable (e.g., when filtering by subject, the system shall provide a list of available subjects instead of requiring manual text input) 2. System shall complete the filter within 2 seconds under normal load condition.
Assumptions:	1. The **“Query School”** page is the main entry point of the application.
Notes and Issues:	TBD: 1. Shall the system allow **human text input** (e.g., “a school near Area A offering subjects B and C”)? a. To be resolved by: Yu Wen Hao b. Due date: 17 September 2025

1.2. UCD

This section describes all use cases related to retrieving and displaying details of a result. All use case IDs in this section are prefixed with **UCD**.

1.2.1 UCD-1 View School Details

Use Case ID:	UCD-1		
Use Case Name:	View School Details		
Created By:	Yong Chee Seng	Last Updated By:	
Date Created:	4 September 2025	Date Last Updated:	

Actor:	User (Initiating)
Description:	User shall be able to view the details of a selected school, and system shall display the full information of that school.
Preconditions:	1. System has received the identifier of the selected school (e.g., school name or unique ID).
Postconditions:	1. System displays the full details of the selected school. 2. System displays any available user comments associated with the school.
Priority:	High
Frequency of Use:	Estimated 100–500 per day across all users
Flow of Events:	1. User selects a school from the query results (or other entry points). 2. System initiates **UCSY-1 Fetch School Data from Government API** to retrieve the details of the selected school using its identifier. 3. System displays the full details of the selected school. 4. System retrieves users' comments related to the school. 5. System displays the retrieved user comments.
Alternative Flows:	AFS3a: **UCSY-1 Throws Exception**

	1. System displays error “Unexpected error occurred. Please try again”. AFS3b: **School Not Found** 1. If school with the identifier is not found, system displays error “School not found”. AFS5: **Comments Unavailable** 1. If comments cannot be retrieved, system displays error “Comments unavailable at the moment”.
Exceptions:	NA
Includes:	1. UCSY-1 Fetch School Data from Government API
Special Requirements:	1. System shall show the details within 2 seconds under normal load condition.
Assumptions:	NA
Notes and Issues:	NA

1.2.2 UCD-2 Compare School

Use Case ID:	UCD-2		
Use Case Name:	Compare School		
Created By:	Yong Chee Seng	Last Updated By:	
Date Created:	4 September 2025	Date Last Updated:	

Actor:	User (Initiating)
Description:	User shall be able to view and compare the details of 2 selected schools side by side.
Preconditions:	1. System has received the identifier of the 2 selected schools (e.g., school name or unique ID).
Postconditions:	1. System displays the full details of the 2 selected schools side by side. 2. System displays any available user comments associated with the 2 selected schools.
Priority:	Medium
Frequency of Use:	Estimated 10–50 per day across all users
Flow of Events:	1. User selects 2 schools from the query results (or other entry points). 2. System initiates **UCSY-1 Fetch School Data from Government API** to retrieve the details of the 2 selected schools using their identifier. 3. System displays the full details of the 2 selected schools side by side. 4. System retrieves users' comments related to the 2 selected schools. 5. System displays the retrieved user comments.
Alternative Flows:	AFS3a: **UCSY-1 Throws Exception** 2. System displays error "Unexpected error occurred. Please try again".

	AFS3b: **School Not Found** 2. If school with the identifier is not found, system displays error “School not found”. AFS5: **Comments Unavailable** 2. If comments cannot be retrieved, system displays error “Comments unavailable at the moment”.
Exceptions:	NA
Includes:	1. UCSY-1 Fetch School Data from Government API
Special Requirements:	1. System shall show the details within 2 seconds under normal load condition. 2. System shall reuse the UI components of **UCD-1 View School Details** where applicable to ensure consistency and maintainability. 3. System shall present the comparison of both schools in a responsive layout that avoids UI overlay issues or excessive horizontal scrolling across all supported devices.
Assumptions:	NA
Notes and Issues:	NA

1.3. UCA

This section describes all use cases related to authentication. All use case IDs in this section are prefixed with **UCA**.

1.3.1 UCA-1 Login

Use Case ID:	UCA-1		
Use Case Name:	Login		
Created By:	Yong Chee Seng	Last Updated By:	
Date Created:	4 September 2025	Date Last Updated:	

Actor:	User (Initiating)
Description:	User shall be able to login to their account by providing valid credentials.
Preconditions:	1. User has signed up an account.
Postconditions:	1. System recognizes the user as authenticated and grants access to logged-in user functionalities. 2. System displays the account username (or equivalent identifier) in the UI.
Priority:	Medium
Frequency of Use:	Estimated 5-10 per day across all users
Flow of Events:	1. User navigates to "Login" page. 2. User inputs username and password. 3. System retrieves the user account information corresponding to the entered username. 4. If user exists, system validates the entered password against the stored password. 5. If password is correct, system creates a session for the authenticated user. 6. System displays message "Login successfully".

	7. System redirects the user to the last page visited (or to the home page if no prior page exists).
Alternative Flows:	AFS4a: **User Not Found** 1. If the entered username does not exist, system displays error “Invalid username”. AFS4b: **User Unavailable** 1. If system cannot access user data, it displays error “Login unavailable. Please try again later”. AFS5: **Incorrect Password** 1. If the password is incorrect, system displays error “Invalid password”.
Exceptions:	NA
Includes:	NA
Special Requirements:	1. System shall complete the login process within 5 seconds under normal load conditions. 2. System shall protect against brute-force login attempts (e.g., account lockout, reCAPTCHA).
Assumptions:	NA
Notes and Issues:	NA

1.3.2 UCA-2 Signup

Use Case ID:	UCA-2		
Use Case Name:	Signup		
Created By:	Yong Chee Seng	Last Updated By:	
Date Created:	4 September 2025	Date Last Updated:	

Actor:	User (Initiating)
Description:	User shall be able to sign up an account by providing valid credentials.
Preconditions:	NA

Postconditions:	1. System stores user credentials securely. 2. System allow user to login with the provided credentials
Priority:	Medium
Frequency of Use:	Estimated 3-5 per day across all users
Flow of Events:	1. User navigates to “Signup” page. 2. User inputs username, password, and confirm password. 3. System checks whether the username is already taken. 4. If the username is available, the system validates that the password meets the security policy (e.g., minimum length of 8 characters, includes uppercase, lowercase, number, and special symbol). 5. If password is secure, system verifies that the password and confirm password match. 6. If the confirm password is same as confirm password, system stores user credentials. 7. The system displays message “Account created. You can login now”. 8. System redirects the user to the “Login” page.
Alternative Flows:	AFS4a: **Username Taken** 1. If the entered username does not exist, system displays error “Username already exists. Please choose another”. AFS4b: **User Unavailable** 1. If system cannot access user data, system displays error “Signup unavailable. Please try again later”. AFS5: **Weak Password** 1. If the password does not meet the security policy, system displays error: “Password does not meet security requirements”. AFS6: **Password Mismatch**

	1. If the password and confirm password do not match, system displays the error: "Passwords do not match". AFS7: **System Error** 1. If credentials cannot be stored due to a system error, system displays error: "Signup unavailable. Please try again later".
Exceptions:	NA
Includes:	NA
Special Requirements:	1. System shall enforce a strong password policy (e.g., minimum length of 8 characters, including uppercase, lowercase, numbers, and special symbols). 2. System shall complete the signup process within 5 seconds under normal load conditions. 3. System shall protect against automated/bot signups (e.g., reCAPTCHA or an equivalent). 4. System shall store all user passwords using industry-standard encryption methods (e.g., bcrypt). 5. The system shall ensure that users explicitly agree to the **Terms and Conditions** and **Privacy Policy** in a clear and visible manner before account creation.
Assumptions:	NA
Notes and Issues:	NA

1.4. UCS

This section describes all use cases related to creating, modifying, and storing data in the system. All use case IDs in this section are prefixed with **UCS**.

1.4.1 UCS-1 Create Comment

Use Case ID:	UCS-1		
Use Case Name:	Create Comment		
Created By:	Yong Chee Seng	Last Updated By:	
Date Created:	4 September 2025	Date Last Updated:	

Actor:	User (Initiating)
Description:	User shall be able to create and submit a comment under any "School Details" page.
Preconditions:	1. User has logged in to an account. 2. User has navigated to a valid "School Details" page.
Postconditions:	1. System stores the submitted comment. 2. System displays the new comment under the related school's comments section.
Priority:	Low
Frequency of Use:	Estimated 3-5 per day across all users
Flow of Events:	1. User navigates to "School Details" page. 2. User inputs comment into the comment input field. 3. User submits the comment. 4. System validates the input (e.g., not empty, within allowed length). 5. If valid, system stores the comment. 6. System refreshes or updates the page (e.g., via AJAX or equivalent). 7. System displays the newly created comment in the comments section.
Alternative Flows:	AFS2: **User Not Logged In** 1. System displays message "You need to login to comment".

	AFS5: **Invalid Input** 1. System displays a clear error message explaining why input is invalid (e.g., empty or too long). AFS6: **System Error** 1. If comment cannot be stored due to a system error, system displays error: “Comment unavailable. Please try again later”.
Exceptions:	NA
Includes:	UCA-1 Login
Special Requirements:	1. System shall complete the comment creation process within 5 seconds under normal load conditions. 2. System shall protect against automated/bot attack (e.g., reCAPTCHA or an equivalent). 3. System shall sanitize and validate user input to prevent malicious attacks (e.g., XSS, SQL injection).
Assumptions:	NA
Notes and Issues:	NA

1.4.2 UCS-2 Reply Comment

Use Case ID:	UCS-2		
Use Case Name:	Reply Comment		
Created By:	Yong Chee Seng	Last Updated By:	
Date Created:	4 September 2025	Date Last Updated:	

Actor:	User (Initiating)
Description:	User shall be able to reply to a comment under any “School Details” page.
Preconditions:	1. User has logged in to an account. 2. User has navigated to a valid “School Details” page.
Postconditions:	1. System stores the submitted reply. 2. System displays the new reply under the related comments.

Priority:	Low
Frequency of Use:	Estimated 3-5 per day across all users
Flow of Events:	1. User navigates to "School Details" page. 2. User selects to reply to a comment. 3. User inputs reply into the input field. 4. User submits the reply. 5. System validates the input (e.g., not empty, within allowed length). 6. If valid, system stores the reply. 7. System refreshes or updates the page (e.g., via AJAX or equivalent). 8. System displays the newly created reply under the related comment.
Alternative Flows:	AFS3: **User Not Logged In** 1. System displays message "You need to login to reply". AFS6: **Invalid Input** 1. System displays a clear error message explaining why input is invalid (e.g., empty or too long). AFS7: **System Error** 1. If reply cannot be stored due to a system error, system displays error: "Reply unavailable. Please try again later".
Exceptions:	NA
Includes:	UCA-1 Login
Special Requirements:	1. System shall complete the reply creation process within 5 seconds under normal load conditions. 2. System shall protect against automated/bot attack (e.g., reCAPTCHA or an equivalent). 3. System shall sanitize and validate user input to prevent malicious attacks (e.g., XSS, SQL injection).
Assumptions:	NA
Notes and Issues:	NA

1.4.3 UCS-3 Vote Comment

Use Case ID:	UCS-3		
Use Case Name:	Vote Comment		
Created By:	Yong Chee Seng	Last Updated By:	
Date Created:	4 September 2025	Date Last Updated:	

Actor:	User (Initiating)
Description:	User shall be able to upvotes or downvotes a comment under any “School Details” page.
Preconditions:	1. User has logged in to an account. 2. User has navigated to a valid “School Details” page.
Postconditions:	1. System stores the submitted upvote or downvote. 2. System displays the new vote count under the related comment.
Priority:	Low
Frequency of Use:	Estimated 5-10 per day across all users
Flow of Events:	1. User navigates to “School Details” page. 2. User selects a comment to upvote or downvote. 3. System stores the vote. 4. System refreshes or updates the page (e.g., via AJAX or equivalent). 5. System displays the new upvote and downvote count under the related comments.
Alternative Flows:	AFS3a: **User Not Logged In** 1. System displays message “You need to login to upvote or downvote”. AFS3b: **User Already Voted** 1. If the user has already voted the comment, the system toggles the vote.

	AFS4: **System Error** 1. If reply cannot be stored due to a system error, system displays error: “Upvote and downvote unavailable. Please try again later”.
Exceptions:	NA
Includes:	UCA-1 Login
Special Requirements:	1. System shall complete the process within 5 seconds under normal load conditions. 2. System shall protect against automated/bot attack (e.g., reCAPTCHA or an equivalent).
Assumptions:	NA
Notes and Issues:	NA

1.4.4 UCS-4 Save School as Favourite

Use Case ID:	UCS-4		
Use Case Name:	Save School as Favourite		
Created By:	Yong Chee Seng	Last Updated By:	
Date Created:	4 September 2025	Date Last Updated:	

Actor:	User (Initiating)
Description:	User shall be able to save a school as favourite.
Preconditions:	1. User has logged in to an account. 2. User has navigated to a valid entry point (e.g., Query School, School Details).
Postconditions:	1. System stores the school as favourite. 2. System visually indicates the favourite status of the school.
Priority:	Medium
Frequency of Use:	Estimated 5-10 per day across all users
Flow of Events:	1. User navigates to valid entry point. 2. User selects a school to save as favourite. 3. System stores the school as favourite for the user. 4. System displays a clear sign the school is saved as favourite (e.g., filled star).
Alternative Flows:	AFS3a: **User Not Logged In** 1. System displays message "You need to login to save the school". AFS3b: **User Already Saved as Favourite** 1. If the user has already saved the school as favourite, the system removes it from favourites and updates the UI indicator. AFS4: **System Error**

	1. If the school cannot be stored or removed due to a system error, system displays: “Unable to update favourite. Please try again later.”
Exceptions:	NA
Includes:	UCA-1 Login
Special Requirements:	1. System shall complete the process within 5 seconds under normal load conditions. 2. System shall protect against automated/bot attack (e.g., reCAPTCHA or an equivalent).
Assumptions:	NA
Notes and Issues:	NA

1.5. UCSY

This section describes all use cases related to system-level actions not directly initiated by the user. All use case IDs in this section are prefixed with **UCSY**.

1.5.1 UCSY-1 Fetch School Data from Government API

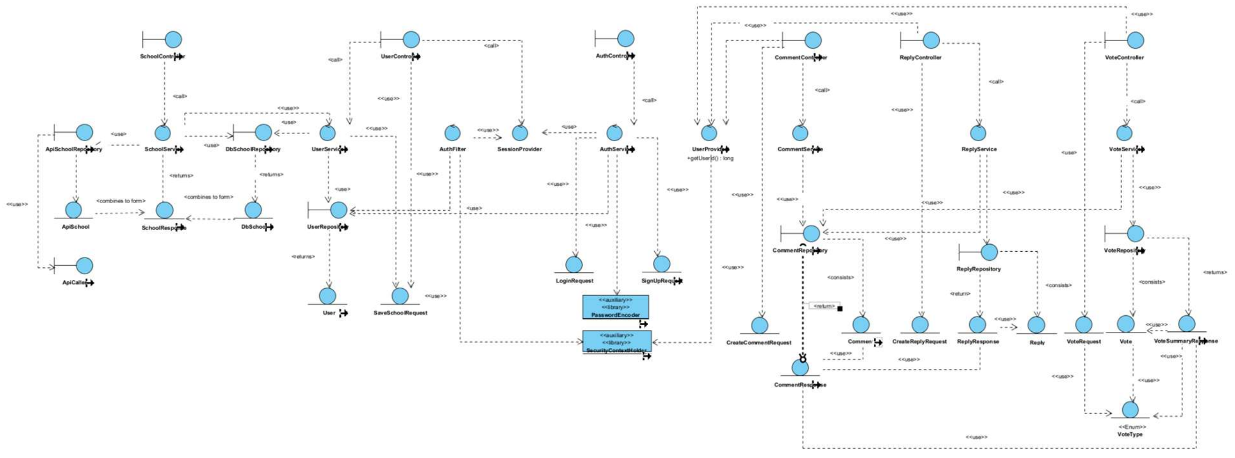
Use Case ID:	UCSY-1		
Use Case Name:	Fetch School Data from Government API		
Created By:	Yong Chee Seng	Last Updated By:	
Date Created:	4 September 2025	Date Last Updated:	

Actor:	Government API
Description:	System shall retrieve available school data from MOE API.
Preconditions:	NA
Postconditions:	1. System caches the retrieved school data. 2. System returns school data when requested.
Priority:	High
Frequency of Use:	Estimated 100-500 per day across all users
Flow of Events:	1. System initiates **UCSY-1 Fetch School Data from Government API.** 2. If school data is not cached, the system calls the MOE API to retrieve all school data. 3. Government API returns all school data. 4. System caches the retrieved school data. 5. If no school identifier is provided, the system returns all cached school data.
Alternative Flows:	AFS2: **School Data is Cached** 1. Continue with step 5. AFS5: **School Identifier is Provided** 1. System returns only the corresponding school data.

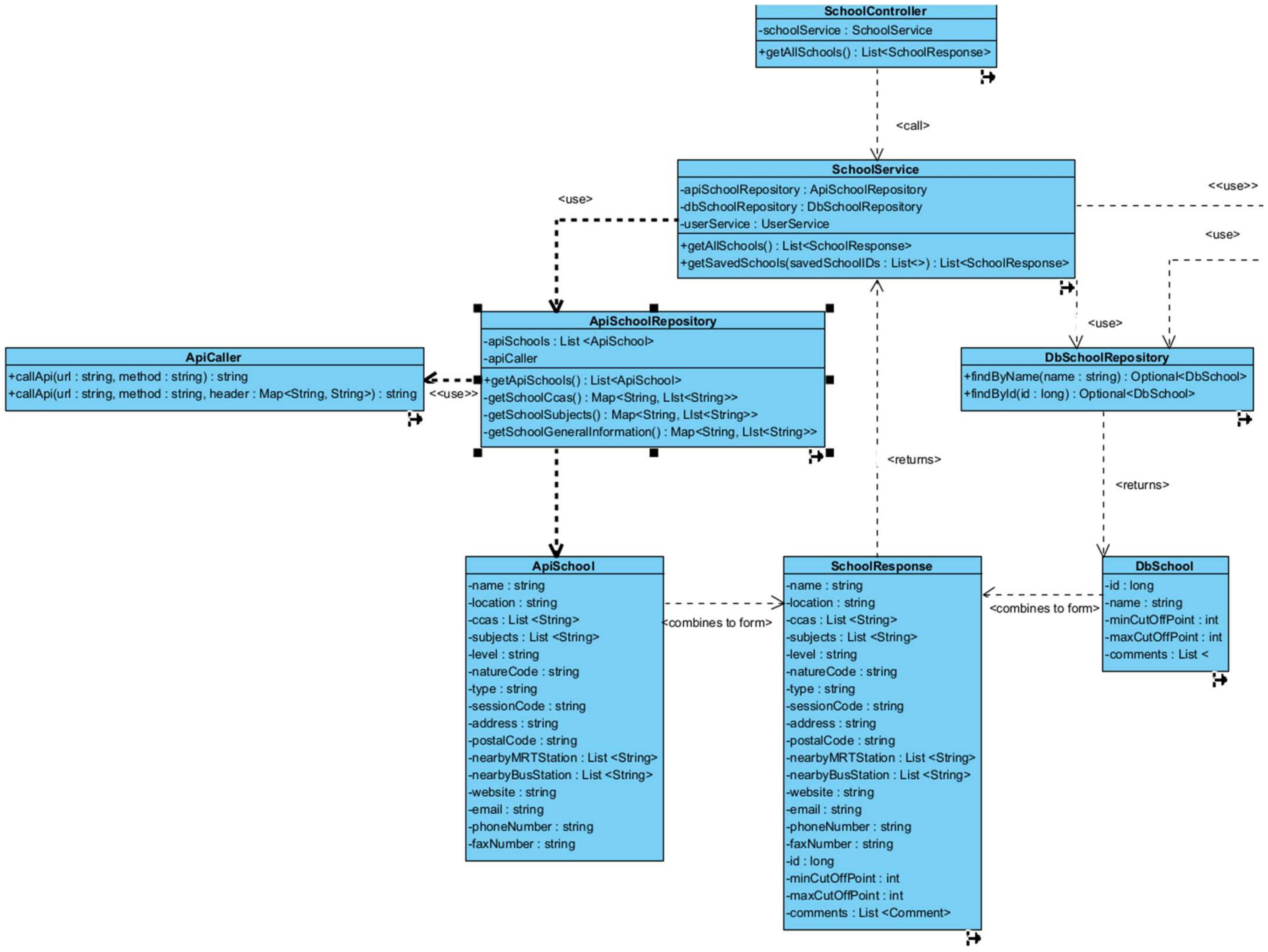
Exceptions:	EX1: **MOE API Unavailable** 1. System throw exception to be caught by initiating process.
Includes:	NA
Special Requirements:	1. System shall complete the process within 5 seconds under normal load conditions. 2. System shall limit external calls to the MOE API to a maximum of 5 times per day.
Assumptions:	NA
Notes and Issues:	NA

C. Class Diagram of Entity Classes

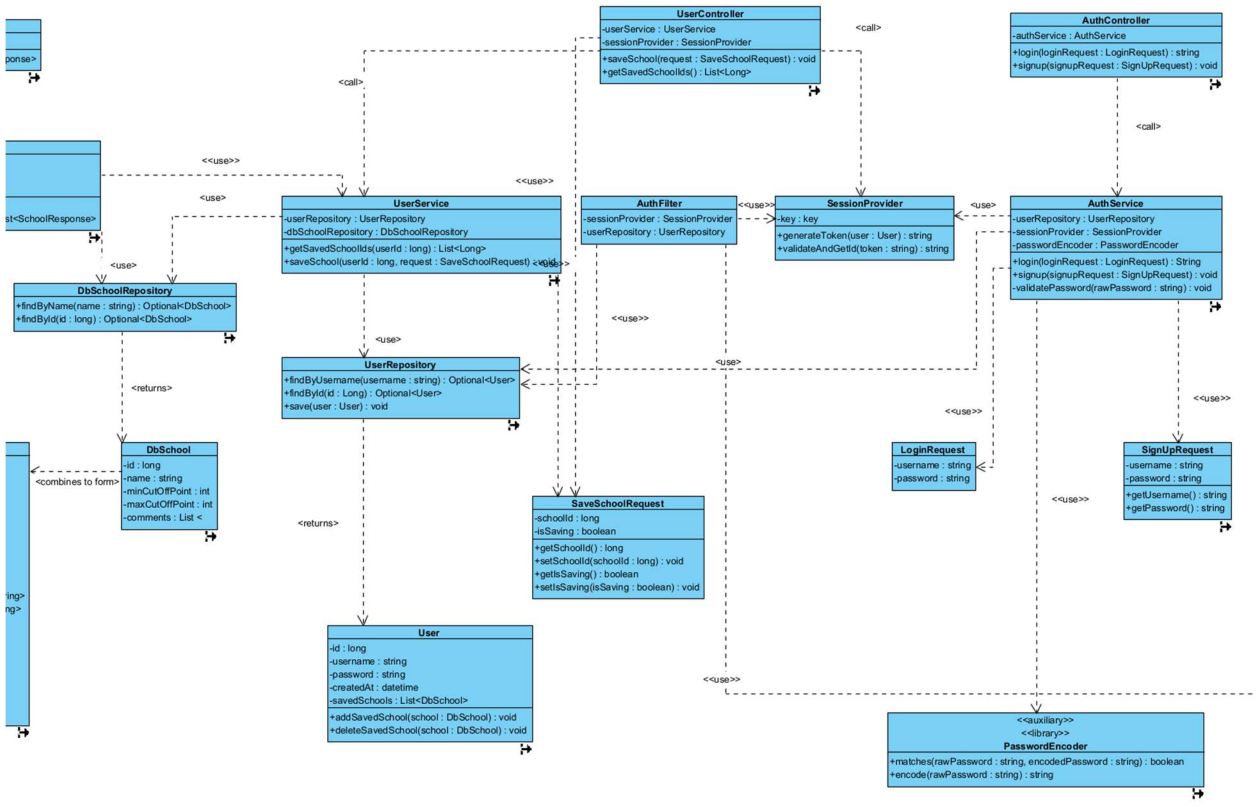
C.1 Stereotype Diagram



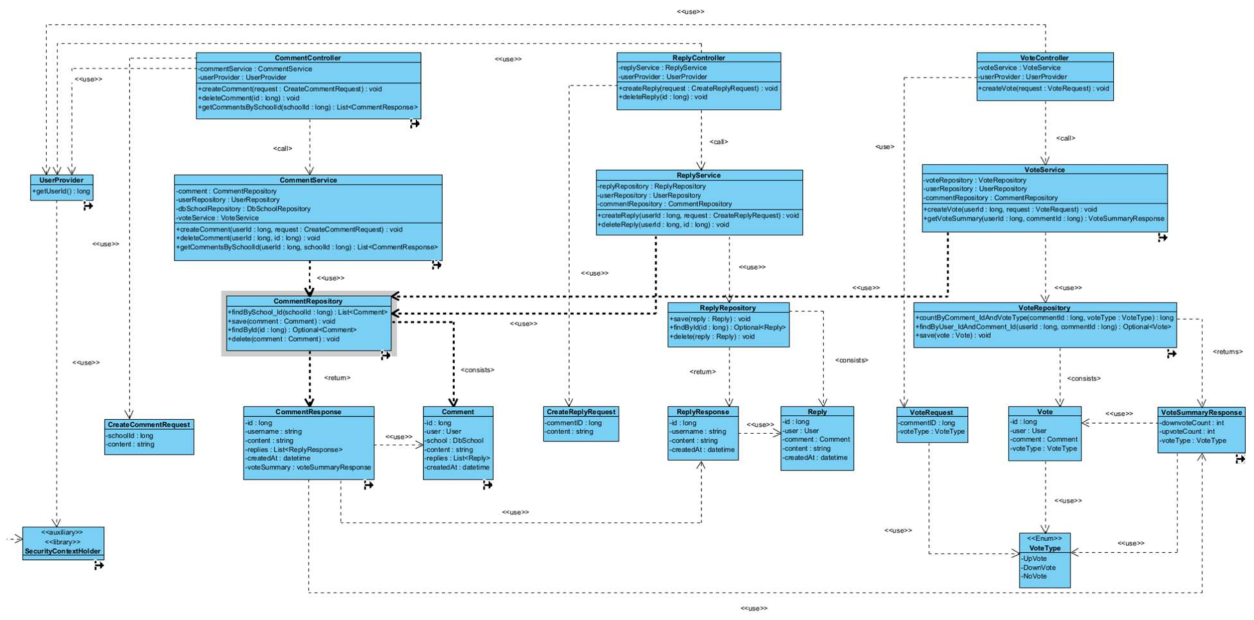
C.2 In-Depth Diagram Of SchoolController


SchoolController
-schoolService : SchoolService
+getAllSchools() : List<SchoolResponse>
<call>
SchoolService
-apiSchoolRepository : ApiSchoolRepository
-dbSchoolRepository : DbSchoolRepository
-userService : UserService
+getAllSchools() : List<SchoolResponse>
+getSavedSchools(savedSchoolIDs : List<>) : List<SchoolResponse>
<<use>>
<use>
<use>
<use>
ApiSchoolRepository
-apiSchools : List <ApiSchool>
-apiCaller
+getApiSchools() : List<ApiSchool>
-getSchoolCcas() : Map<String, LIst<String>>
-getSchoolSubjects() : Map<String, LIst<String>>
-getSchoolGeneralInformation() : Map<String, LIst<String>>
ApiCaller
+callApi(url : string, method : string) : string
+callApi(url : string, method : string, header : Map<String, String>) : string
<<use>>
DbSchoolRepository
+findByName(name : string) : Optional<DbSchool>
+findById(id : long) : Optional<DbSchool>
<returns>
<returns>
ApiSchool
-name : string
-location : string
-ccas : List <String>
-subjects : List <String>
-level : string
-natureCode : string
-type : string
-sessionCode : string
-address : string
-postalCode : string
-nearbyMRTStation : List <String>
-nearbyBusStation : List <String>
-website : string
-email : string
-phoneNumber : string
-faxNumber : string
<combines to form>
SchoolResponse
-name : string
-location : string
-ccas : List <String>
-subjects : List <String>
-level : string
-natureCode : string
-type : string
-sessionCode : string
-address : string
-postalCode : string
-nearbyMRTStation : List <String>
-nearbyBusStation : List <String>
-website : string
-email : string
-phoneNumber : string
-faxNumber : string
-id : long
-minCutOffPoint : int
-maxCutOffPoint : int
-comments : List <Comment>
<combines to form>
DbSchool
-id : long
-name : string
-minCutOffPoint : int
-maxCutOffPoint : int
-comments : List <

C.3 In-Depth Diagram Of UserController & Auth Controller

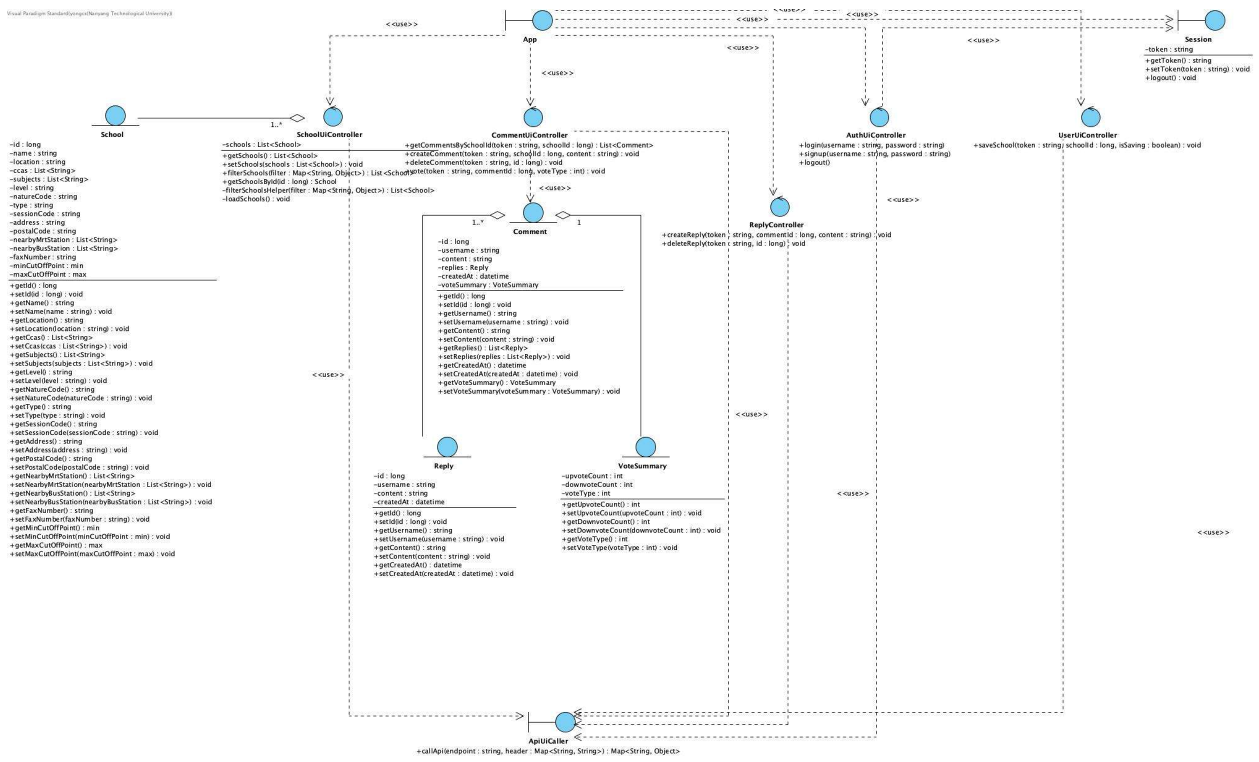

UserController
-userService : UserService
-sessionProvider : SessionProvider
+saveSchool(request : SaveSchoolRequest) : void
+getSavedSchoolIds() : List<Long>
AuthController
-authService : AuthService
+login(loginRequest : LoginRequest) : string
+signup(signupRequest : SignUpRequest) : void
UserService
-userRepository : UserRepository
-dbSchoolRepository : DbSchoolRepository
+getSavedSchoolIds(userId : long) : List<Long>
+saveSchool(userId : long, request : SaveSchoolRequest) : void
AuthFilter
-sessionProvider : SessionProvider
-userRepository : UserRepository
SessionProvider
-key : key
+generateToken(user : User) : string
+validateAndGetId(token : string) : string
AuthService
-userRepository : UserRepository
-sessionProvider : SessionProvider
-passwordEncoder : PasswordEncoder
+login(loginRequest : LoginRequest) : String
+signup(signupRequest : SignUpRequest) : void
-validatePassword(rawPassword : string) : void
DbSchoolRepository
+findByName(name : string) : Optional<DbSchool>
+findById(id : long) : Optional<DbSchool>
UserRepository
+findByUsername(username : string) : Optional<User>
+findById(id : Long) : Optional<User>
+save(user : User) : void
DbSchool
-id : long
-name : string
-minCutOffPoint : int
-maxCutOffPoint : int
-comments : List <
LoginRequest
-username : string
-password : string
SignUpRequest
-username : string
-password : string
+getUsername() : string
+getPassword() : string
SaveSchoolRequest
-schoolId : long
-isSaving : boolean
+getSchoolId() : long
+setSchoolId(schoolId : long) : void
+getIsSaving() : boolean
+setIsSaving(isSaving : boolean) : void
User
-id : long
-username : string
-password : string
-createdAt : datetime
-savedSchools : List<DbSchool>
+addSavedSchool(school : DbSchool) : void
+deleteSavedSchool(school : DbSchool) : void
<<auxiliary>>
<<library>>
PasswordEncoder
+matches(rawPassword : string, encodedPassword : string) : boolean
+encode(rawPassword : string) : string
ponse>
st<SchoolResponse>
ring>
ng>
<call>
<<use>>
<use>
<returns>
<combines to form>

C.6 In-Depth Diagram Of CommentController, ReplyController, VoteController



<<use>>
CommentController
-commentService : CommentService
-userProvider : UserProvider
+createComment(request : CreateCommentRequest) : void
+deleteComment(id : long) : void
+getCommentsBySchoolId(schoolId : long) : List<CommentResponse>
<<use>>
<<use>>
ReplyController
-replyService : ReplyService
-userProvider : UserProvider
+createReply(request : CreateReplyRequest) : void
+deleteReply(id : long) : void
VoteController
-voteService : VoteService
-userProvider : UserProvider
+createVote(request : VoteRequest) : void
<call>
<call>
<use>
<call>
UserProvider
+getUserId() : long
CommentService
-comment : CommentRepository
-userRepository : UserRepository
-dbSchoolRepository : DbSchoolRepository
-voteService : VoteService
+createComment(userId : long, request : CreateCommentRequest) : void
+deleteComment(userId : long, id : long) : void
+getCommentsBySchoolId(userId : long, schoolId : long) : List<CommentResponse>
ReplyService
-replyRepository : ReplyRepository
-userRepository : UserRepository
-commentRepository : CommentRepository
+createReply(userId : long, request : CreateReplyRequest) : void
+deleteReply(userId : long, id : long) : void
VoteService
-voteRepository : VoteRepository
-userRepository : UserRepository
-commentRepository : CommentRepository
+createVote(userId : long, request : VoteRequest) : void
+getVoteSummary(userId : long, commentId : long) : VoteSummaryResponse
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>
CommentRepository
+findBySchool_Id(schoolId : long) : List<Comment>
+save(comment : Comment) : void
+findById(id : long) : Optional<Comment>
+delete(comment : Comment) : void
<<use>>
ReplyRepository
+save(reply : Reply) : void
+findById(id : long) : Optional<Reply>
+delete(reply : Reply) : void
VoteRepository
+countByComment_IdAndVoteType(commentId : long, voteType : VoteType) : long
+findByUser_IdAndComment_Id(userId : long, commentId : long) : Optional<Vote>
+save(vote : Vote) : void
<<use>>
<return>
<consists>
<return>
<consists>
<consists>
<returns>
CreateCommentRequest
-schoolId : long
-content : string
CommentResponse
-id : long
-username : string
-content : string
-replies : List<ReplyResponse>
-createdAt : datetime
-voteSummary : voteSummaryResponse
<<use>>
Comment
-id : long
-user : User
-school : DbSchool
-content : string
-replies : List<Reply>
-createdAt : datetime
CreateReplyRequest
-commentID : long
-content : string
ReplyResponse
-id : long
-username : string
-content : string
-createdAt : datetime
<<use>>
Reply
-id : long
-user : User
-comment : Comment
-content : string
-createdAt : datetime
VoteRequest
-commentID : long
-voteType : VoteType
Vote
-id : long
-user : User
-comment : Comment
-voteType : VoteType
<<use>>
VoteSummaryResponse
-downvoteCount : int
-upvoteCount : int
-voteType : VoteType
<<use>>
<<use>>
<<auxiliary>>
<<library>>
SecurityContextHolder
<<use>>
<<use>>
<<Enum>>
VoteType
-UpVote
-DownVote
-NoVote
<<use>>

C.7 Class Diagram Of Frontend


Visual Paradigm Standard(yongcs(Nanyang Technological University))
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>
App
Session
-token : string
+getToken() : string
+setToken(token : string) : void
+logout() : void
School
-id : long
-name : string
-location : string
-ccas : List<String>
-subjects : List<String>
-level : string
-natureCode : string
-type : string
-sessionCode : string
-address : string
-postalCode : string
-nearbyMrtStation : List<String>
-nearbyBusStation : List<String>
-faxNumber : string
-minCutOffPoint : min
-maxCutOffPoint : max
+getId() : long
+setId(id : long) : void
+getName() : string
+setName(name : string) : void
+getLocation() : string
+setLocation(location : string) : void
+getCcas() : List<String>
+setCcas(ccas : List<String>) : void
+getSubjects() : List<String>
+setSubjects(subjects : List<String>) : void
+getLevel() : string
+setLevel(level : string) : void
+getNatureCode() : string
+setNatureCode(natureCode : string) : void
+getType() : string
+setType(type : string) : void
+getSessionCode() : string
+setSessionCode(sessionCode : string) : void
+getAddress() : string
+setAddress(address : string) : void
+getPostalCode() : string
+setPostalCode(postalCode : string) : void
+getNearbyMrtStation() : List<String>
+setNearbyMrtStation(nearbyMrtStation : List<String>) : void
+getNearbyBusStation() : List<String>
+setNearbyBusStation(nearbyBusStation : List<String>) : void
+getFaxNumber() : string
+setFaxNumber(faxNumber : string) : void
+getMinCutOffPoint() : min
+setMinCutOffPoint(minCutOffPoint : min) : void
+getMaxCutOffPoint() : max
+setMaxCutOffPoint(maxCutOffPoint : max) : void
1..*
SchoolUiController
-schools : List<School>
+getSchools() : List<School>
+setSchools(schools : List<School>) : void
+filterSchools(filter : Map<String, Object>) : List<School>
+getSchoolsById(id : long) : School
-filterSchoolsHelper(filter : Map<String, Object>) : List<School>
-loadSchools() : void
CommentUiController
+getCommentsBySchoolId(token : string, schoolId : long) : List<Comment>
+createComment(token : string, schoolId : long, content : string) : void
+deleteComment(token : string, id : long) : void
+vote(token : string, commentId : long, voteType : int) : void
AuthUiController
+login(username : string, password : string)
+signup(username : string, password : string)
+logout()
UserUiController
+saveSchool(token : string, schoolId : long, isSaving : boolean) : void
ReplyController
+createReply(token : string, commentId : long, content : string) : void
+deleteReply(token : string, id : long) : void
1..*
1
Comment
-id : long
-username : string
-content : string
-replies : Reply
-createdAt : datetime
-voteSummary : VoteSummary
+getId() : long
+setId(id : long) : void
+getUsername() : string
+setUsername(username : string) : void
+getContent() : string
+setContent(content : string) : void
+getReplies() : List<Reply>
+setReplies(replies : List<Reply>) : void
+getCreatedAt() : datetime
+setCreatedAt(createdAt : datetime) : void
+getVoteSummary() : VoteSummary
+setVoteSummary(voteSummary : VoteSummary) : void
Reply
-id : long
-username : string
-content : string
-createdAt : datetime
+getId() : long
+setId(id : long) : void
+getUsername() : string
+setUsername(username : string) : void
+getContent() : string
+setContent(content : string) : void
+getCreatedAt() : datetime
+setCreatedAt(createdAt : datetime) : void
VoteSummary
-upvoteCount : int
-downvoteCount : int
-voteType : int
+getUpvoteCount() : int
+setUpvoteCount(upvoteCount : int) : void
+getDownvoteCount() : int
+setDownvoteCount(downvoteCount : int) : void
+getVoteType() : int
+setVoteType(voteType : int) : void
ApiUiCaller
+callApi(endpoint : string, header : Map<String, String>) : Map<String, Object>

D. Key Boundary Classes and Control Classes

D.1 SchoolController

Method	Description
getAllSchools(): List <SchoolResponse>	Upon loading into the website, this function is triggered and the SchoolController calls the SchoolService

D.2 SchoolService

Method	Description
getAllSchools(): List <SchoolResponse>	SchoolService calls APISchoolRepository, returning the entire list of SchoolResponses to be displayed.
getSavedSchools(savedSchoolsIDs: List<long>): List<SchoolResponse>	SchoolService calls UserService to getSavedSchoolIds and passes this into APISchoolRepository, returning a list of SchoolResponses to be displayed.

D.3 UserController

Method	Description
getSavedSchoolsID(): List <Long>	When a user logs in, the UserController calls the UserService to return the list of schools that the user has saved previously.
saveSchool(request: SaveSchoolRequest): void	When user presses the Save School Button, a SaveSchoolRequest is passed from userController to UserService

D.4 UserService

Method	Description
getSavedSchoolsIds(userId: Long): List<Long>	UserService gets the userId from the SessionProvider and passes into the DbSchoolRepository to obtain the list of previously saved school ids.
saveSchool(userId: long, request: SaveSchoolRequest): void	The UserService obtains the userid from the Session Provider and passes it together with the SaveSchoolRequest to the DbSchoolRepository to save the school.

D.5 SessionProvider

Method	Description
generateToken(user: User): string	Builds a signed token containing subject (username/user id) and returns it
validateAndGetId(token: string): long	Verifies token and returns user_id if valid, if not it throws an exception

D.6 AuthController

Method	Description
login(loginRequest: LoginRequest)	Passes a loginRequest object to AuthService after user attemps to login
signup(signupRequest:SignupRequest)	Passes a loginRequest object to AuthService after user signs up

D.7 AuthService

Method	Description
login(loginRequest: LoginRequest)	Verifies user via sessionprovider
signup(signupRequest:SignupRequest)	Creates new user in user database
validatePassword(rawPassword: string): void	A private method that is only called by login to help validate the password in the LoginRequest

D.8 CommentController

Method	Description
createComment(createCommentRequest: CreateCommentRequest): void	When user writes a comment and sends it, a CreateCommentRequest object is sent to CommentService.
deleteComment(deleteCommentRequest: DeleteCommentRequest): void	When user presses the delete comment button, a DeleteCommentRequest object is sent to CommentService.
getCommentsBySchoolId(schoolid: Long): List<CommentResponse>	When user loads into a school, the schoolid is sent to CommentService.

D.9 CommentService

Method	Description
createComment(userId: long, createCommentRequest: CreateCommentRequest): void	CommentService accepts CreateCommentRequest object from CommentController and sends it to commentRepository to add it to the comment database.
deleteComment(userId: long, deleteCommentRequest: DeleteCommentRequest): void	CommentService accepts DeleteCommentRequest object from CommentController and sends it to commentRepository to delete the comment from the database.
getCommentsBySchoolId(schoolid: Long): List<CommentResponse>	CommentService accepts schoolid from CommentController and searches up the comments in commentRepository, returning a list of CommentResponses.

D.11 ReplyController

Method	Description
createReply(createReplyRequest: CreateReplyRequest): void	When user replies to a comment and sends it, a CreateReplyRequest object is sent to CommentService.
deleteReply(id: long): void	When user presses the delete replybutton, the reply's ID will be passed to ReplyService.

D.12 ReplyService

Method	Description
createReply(userId: long, createReplyRequest: CreateReplyRequest): void	ReplyService accepts CreateReplyRequest object from ReplyController and sends it to ReplyRepository to add it to the comment database.
deleteReply(userId: long, id:long): void	ReplyService accepts reply_id from ReplyController and sends it to ReplyRepository to delete the reply.

D.13 VoteController

Method	Description
createVote(userId: long, voteRequest: VoteRequest): void	When user upvotes or downvotes or removes his vote, a VoteRequest object is sent to VoteService.

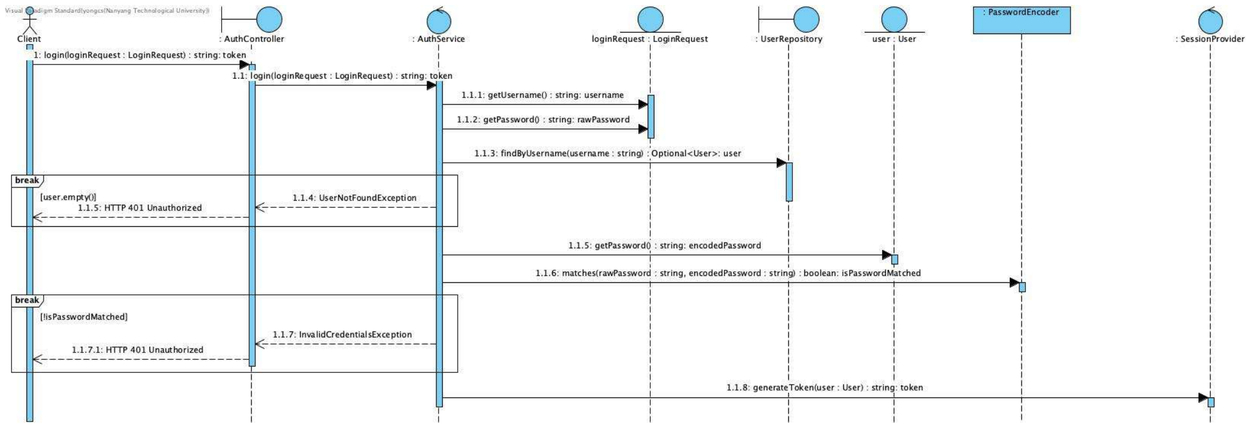
D.14 VoteService

Method	Description
createVote(userId: long, request:VoteRequest):void	VoteService accepts VoteRequest from VoteController and sends it to VoteRepository to recalculate total votes for the comment.
getVotesByCommentId(userId: long, commentId: long): VoteSummaryResponse	VoteService looks up VoteSummaryResponse from VoteRepository and returns it.

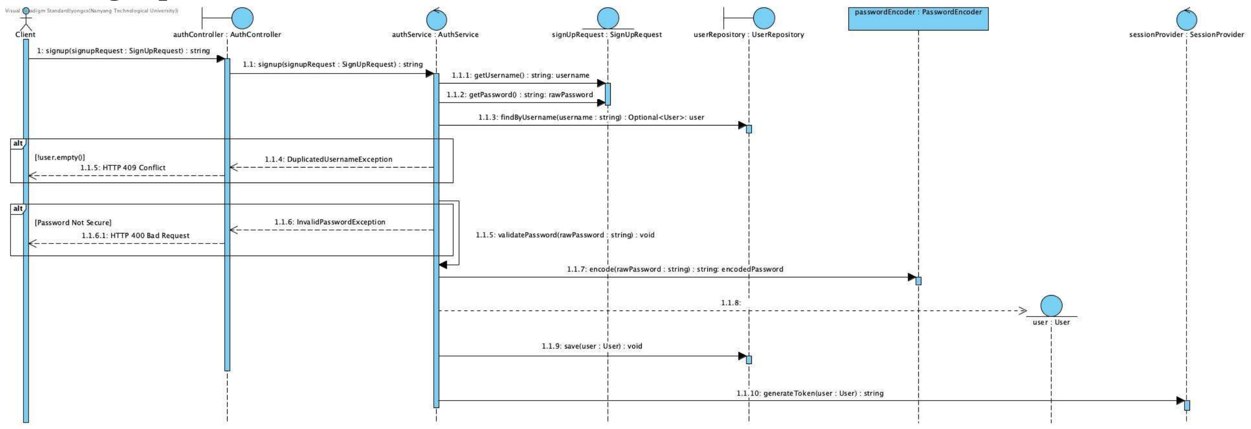
E. Sequence Diagrams of Use Cases

<u>Backend</u>

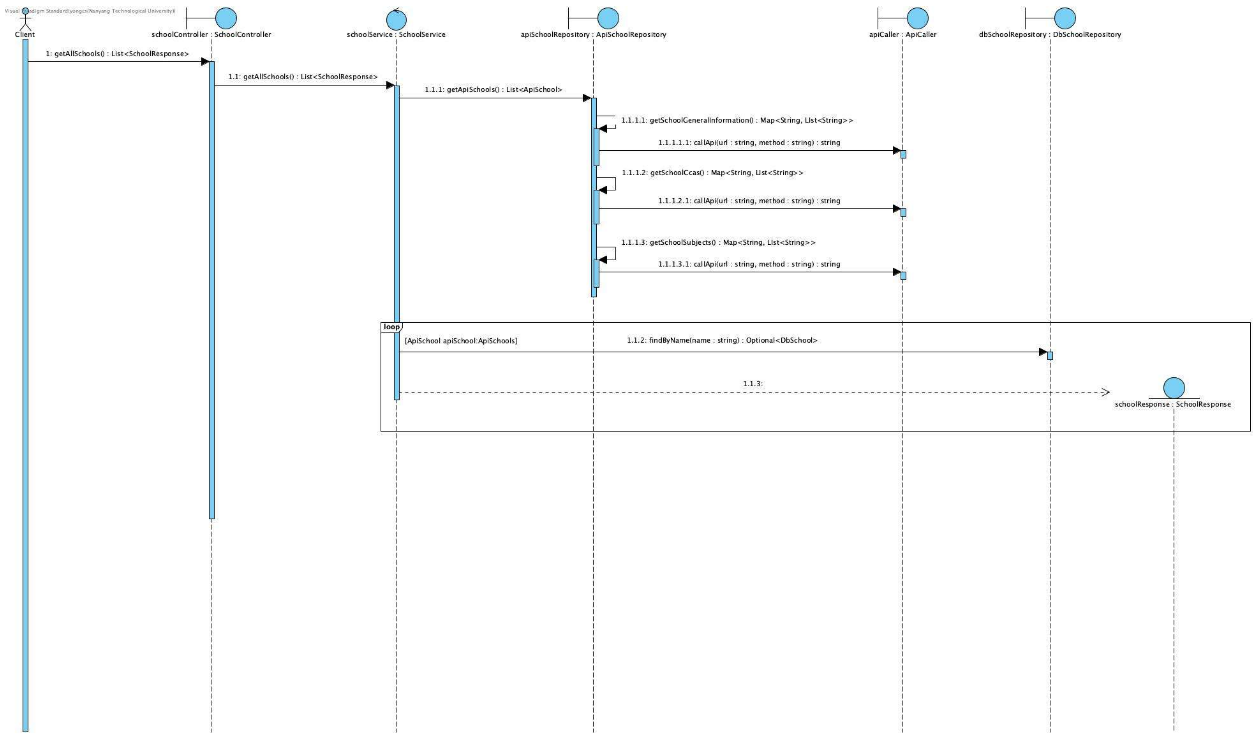
EB.1 Login


Visual Paradigm Standard(yongcs(Nanyang Technological University))
Client
: AuthController
: AuthService
loginRequest : LoginRequest
: UserRepository
user : User
: PasswordEncoder
: SessionProvider
1: login(loginRequest : LoginRequest) : string: token
1.1: login(loginRequest : LoginRequest) : string: token
1.1.1: getUsername() : string: username
1.1.2: getPassword() : string: rawPassword
1.1.3: findByUsername(username : string) : Optional<User>: user
break
[user.empty()]
1.1.4: UserNotFoundException
1.1.5: HTTP 401 Unauthorized
1.1.5: getPassword() : string: encodedPassword
1.1.6: matches(rawPassword : string, encodedPassword : string) : boolean: isPasswordMatched
break
[!isPasswordMatched]
1.1.7: InvalidCredentialsException
1.1.7.1: HTTP 401 Unauthorized
1.1.8: generateToken(user : User) : string: token

EB.2 Signup

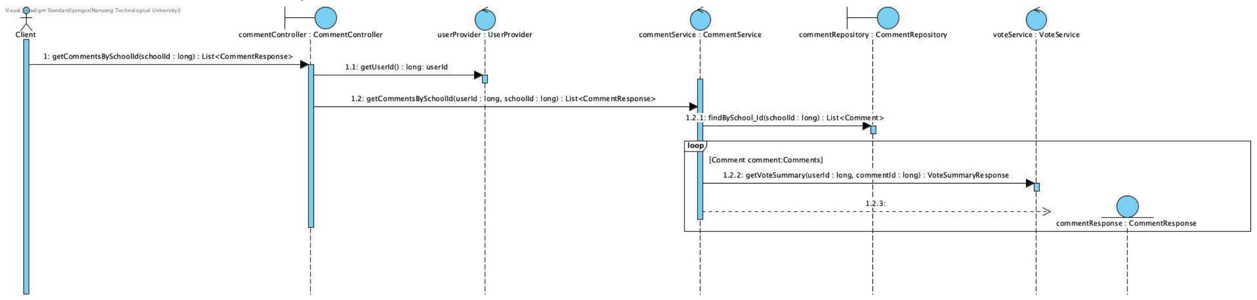

Visual Paradigm Standard(yongcs(Nanyang Technological University))
Client
authController : AuthController
authService : AuthService
signUpRequest : SignUpRequest
userRepository : UserRepository
passwordEncoder : PasswordEncoder
sessionProvider : SessionProvider
1: signup(signupRequest : SignUpRequest) : string
1.1: signup(signupRequest : SignUpRequest) : string
1.1.1: getUsername() : string: username
1.1.2: getPassword() : string: rawPassword
1.1.3: findByUsername(username : string) : Optional<User>: user
alt
[!user.empty()]
1.1.4: DuplicatedUsernameException
1.1.5: HTTP 409 Conflict
alt
[Password Not Secure]
1.1.6: InvalidPasswordException
1.1.6.1: HTTP 400 Bad Request
1.1.5: validatePassword(rawPassword : string) : void
1.1.7: encode(rawPassword : string) : string: encodedPassword
1.1.8:
user : User
1.1.9: save(user : User) : void
1.1.10: generateToken(user : User) : string

EB.3 GetSchools



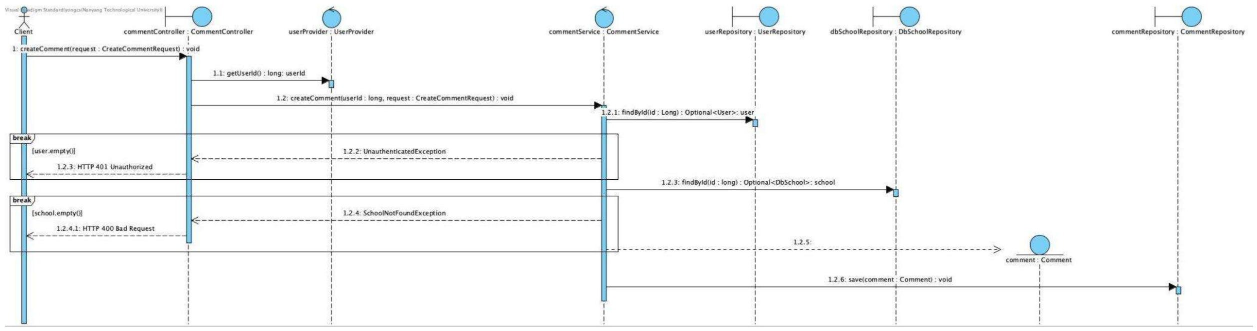
Visual Paradigm Standard(yongcs(Nanyang Technological University))
Client
schoolController : SchoolController
schoolService : SchoolService
apiSchoolRepository : ApiSchoolRepository
apiCaller : ApiCaller
dbSchoolRepository : DbSchoolRepository
1: getAllSchools() : List<SchoolResponse>
1.1: getAllSchools() : List<SchoolResponse>
1.1.1: getApiSchools() : List<ApiSchool>
1.1.1.1: getSchoolGeneralInformation() : Map<String, List<String>>
1.1.1.1.1: callApi(url : string, method : string) : string
1.1.1.2: getSchoolCcas() : Map<String, List<String>>
1.1.1.2.1: callApi(url : string, method : string) : string
1.1.1.3: getSchoolSubjects() : Map<String, List<String>>
1.1.1.3.1: callApi(url : string, method : string) : string
loop
[ApiSchool apiSchool:ApiSchools]
1.1.2: findByName(name : string) : Optional<DbSchool>
1.1.3:
schoolResponse : SchoolResponse

EB.4 GetCommentsBySchoolId



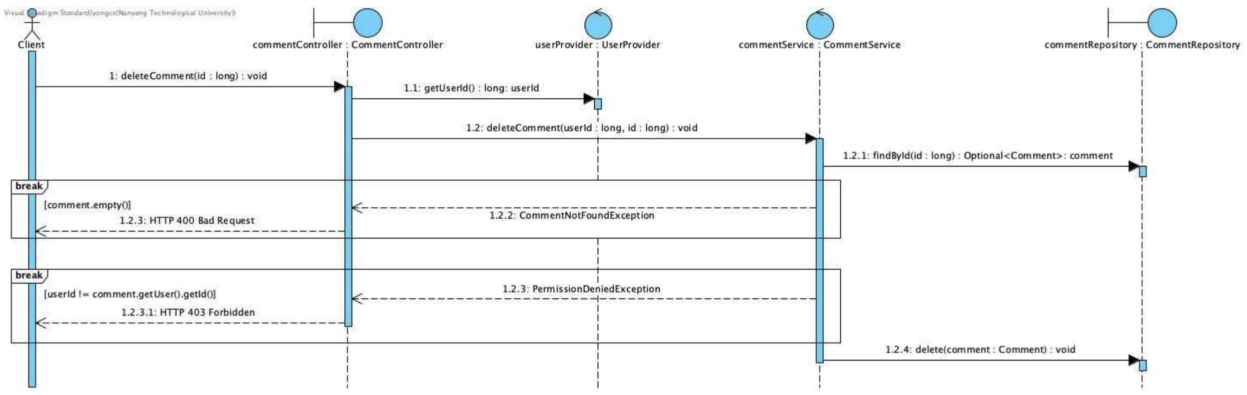
Visual Paradigm Standard(yongcs(Nanyang Technological University))
Client
commentController : CommentController
userProvider : UserProvider
commentService : CommentService
commentRepository : CommentRepository
voteService : VoteService
1: getCommentsBySchoolId(schoolId : long) : List<CommentResponse>
1.1: getUserId() : long: userId
1.2: getCommentsBySchoolId(userId : long, schoolId : long) : List<CommentResponse>
1.2.1: findBySchool_Id(schoolId : long) : List<Comment>
loop
[Comment comment:Comments]
1.2.2: getVoteSummary(userId : long, commentId : long) : VoteSummaryResponse
1.2.3:
commentResponse : CommentResponse

EB.5 CreateComment

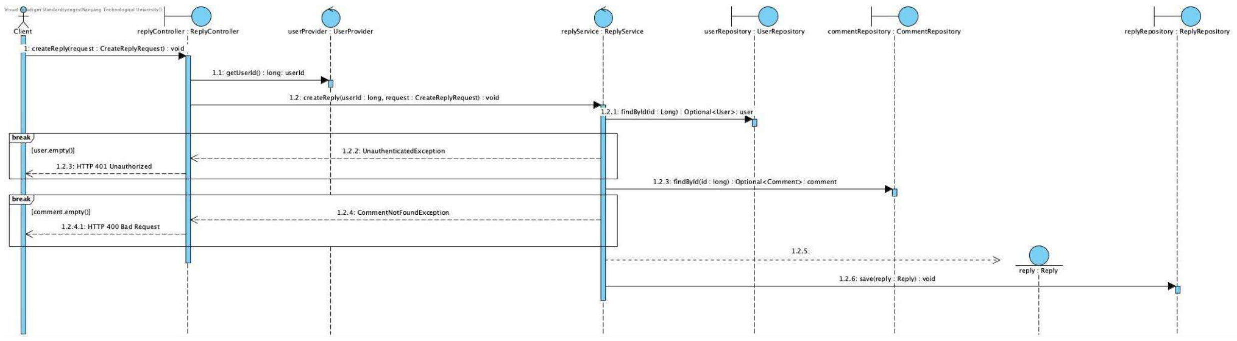


Visual Paradigm Standard(yongcs(Nanyang Technological University))
Client
commentController : CommentController
userProvider : UserProvider
commentService : CommentService
userRepository : UserRepository
dbSchoolRepository : DbSchoolRepository
commentRepository : CommentRepository
1: createComment(request : CreateCommentRequest) : void
1.1: getUserId() : long: userId
1.2: createComment(userId : long, request : CreateCommentRequest) : void
1.2.1: findById(id : Long) : Optional<User>: user
break
[user.empty()]
1.2.2: UnauthenticatedException
1.2.3: HTTP 401 Unauthorized
1.2.3: findById(id : long) : Optional<DbSchool>: school
break
[school.empty()]
1.2.4: SchoolNotFoundException
1.2.4.1: HTTP 400 Bad Request
1.2.5:
comment : Comment
1.2.6: save(comment : Comment) : void

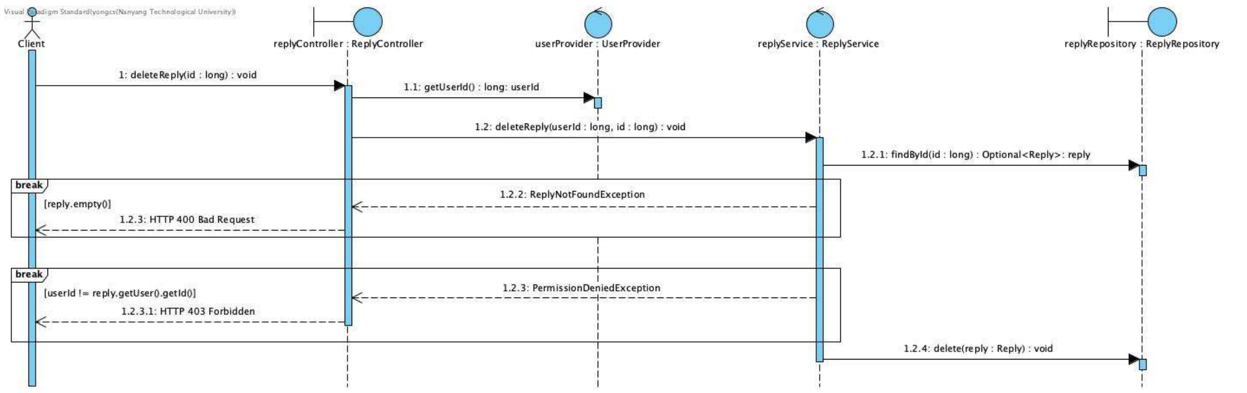
EB.6 DeleteComment


Visual Paradigm Standard(yongcs(Nanyang Technological University))
Client
commentController : CommentController
userProvider : UserProvider
commentService : CommentService
commentRepository : CommentRepository
1: deleteComment(id : long) : void
1.1: getUserId() : long: userId
1.2: deleteComment(userId : long, id : long) : void
1.2.1: findById(id : long) : Optional<Comment>: comment
break
[comment.empty()]
1.2.2: CommentNotFoundException
1.2.3: HTTP 400 Bad Request
break
[userId != comment.getUser().getId()]
1.2.3: PermissionDeniedException
1.2.3.1: HTTP 403 Forbidden
1.2.4: delete(comment : Comment) : void

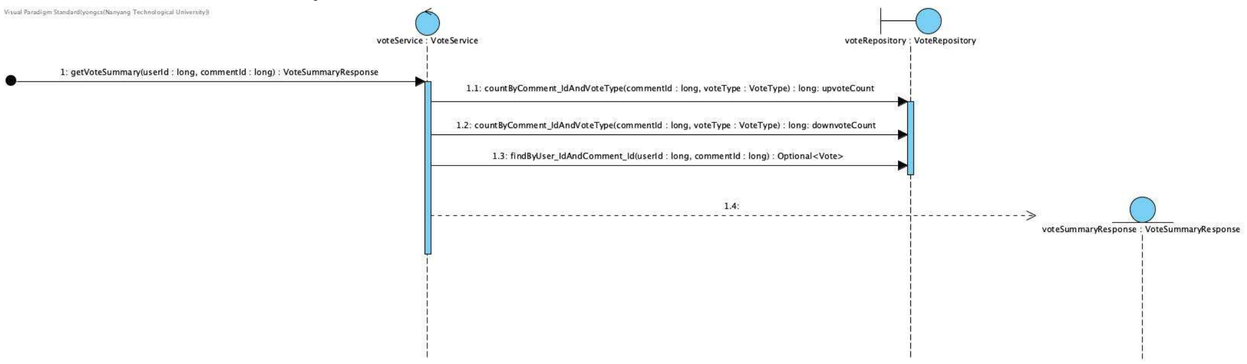
EB.7 CreateReply


Visual Paradigm Standard(yongcs(Nanyang Technological University))
Client
replyController : ReplyController
userProvider : UserProvider
replyService : ReplyService
userRepository : UserRepository
commentRepository : CommentRepository
replyRepository : ReplyRepository
1: createReply(request : CreateReplyRequest) : void
1.1: getUserId() : long: userId
1.2: createReply(userId : long, request : CreateReplyRequest) : void
1.2.1: findById(id : Long) : Optional<User>: user
break
[user.empty()]
1.2.2: UnauthenticatedException
1.2.3: HTTP 401 Unauthorized
1.2.3: findById(id : long) : Optional<Comment>: comment
break
[comment.empty()]
1.2.4: CommentNotFoundException
1.2.4.1: HTTP 400 Bad Request
1.2.5:
reply : Reply
1.2.6: save(reply : Reply) : void

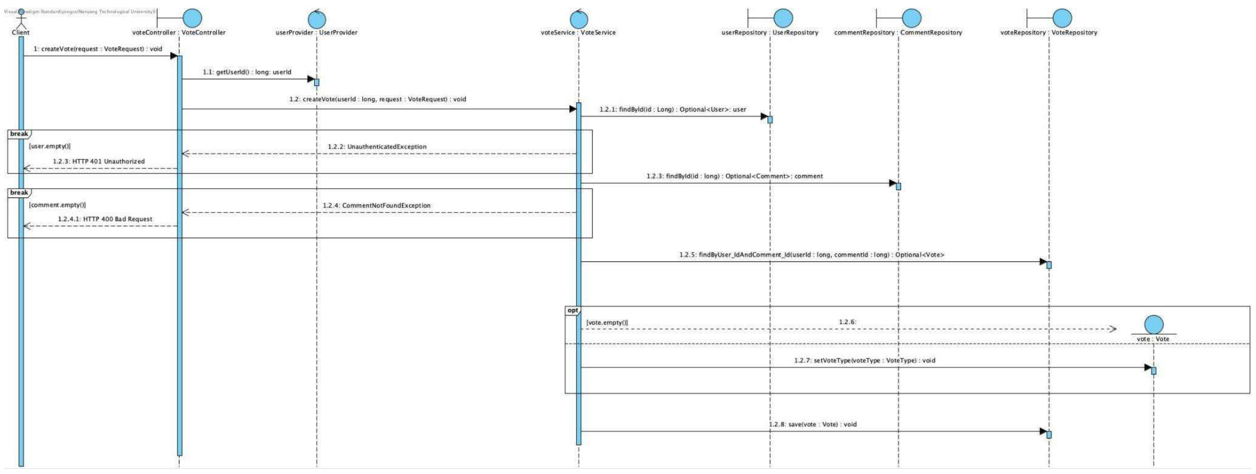
EB.8 DeleteReply


Visual Paradigm Standard(yongcs(Nanyang Technological University))
Client
replyController : ReplyController
userProvider : UserProvider
replyService : ReplyService
replyRepository : ReplyRepository
1: deleteReply(id : long) : void
1.1: getUserId() : long: userId
1.2: deleteReply(userId : long, id : long) : void
1.2.1: findById(id : long) : Optional<Reply>: reply
break
[reply.empty()]
1.2.2: ReplyNotFoundException
1.2.3: HTTP 400 Bad Request
break
[userId != reply.getUser().getId()]
1.2.3: PermissionDeniedException
1.2.3.1: HTTP 403 Forbidden
1.2.4: delete(reply : Reply) : void

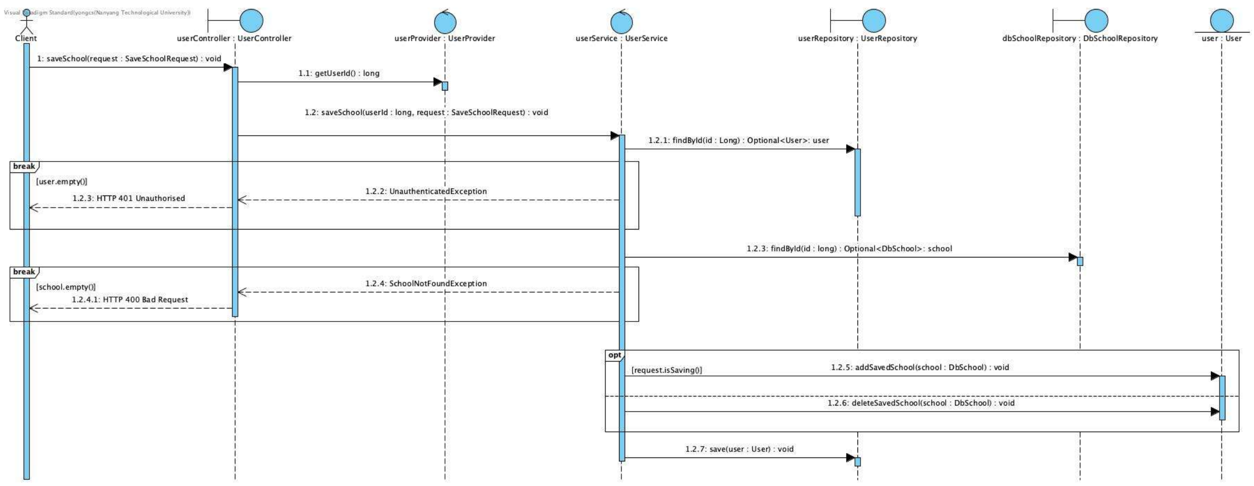
EB.9 GetVoteSummary


Visual Paradigm Standard(yongcs(Nanyang Technological University))
voteService : VoteService
voteRepository : VoteRepository
1: getVoteSummary(userId : long, commentId : long) : VoteSummaryResponse
1.1: countByComment_IdAndVoteType(commentId : long, voteType : VoteType) : long: upvoteCount
1.2: countByComment_IdAndVoteType(commentId : long, voteType : VoteType) : long: downvoteCount
1.3: findByUser_IdAndComment_Id(userId : long, commentId : long) : Optional<Vote>
1.4:
voteSummaryResponse : VoteSummaryResponse

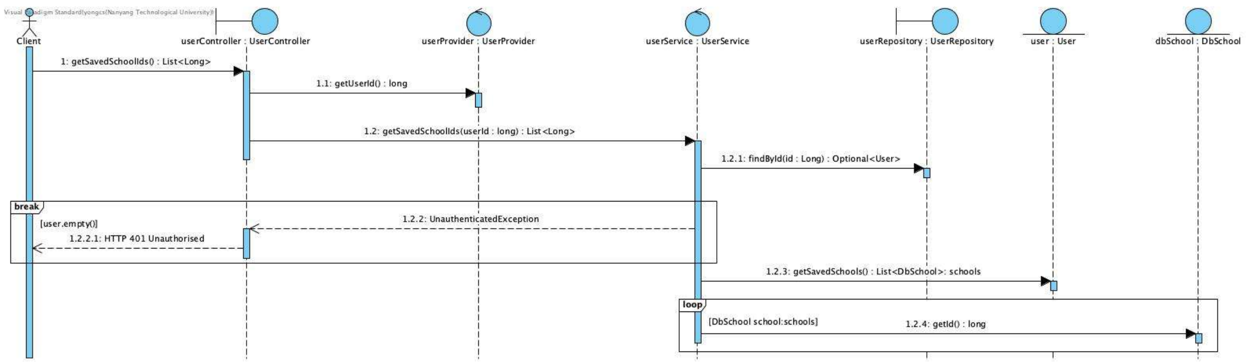
EB.10 CreateVote


Visual Paradigm Standard(yongcs(Nanyang Technological University))
Client
voteController : VoteController
userProvider : UserProvider
voteService : VoteService
userRepository : UserRepository
commentRepository : CommentRepository
voteRepository : VoteRepository
1: createVote(request : VoteRequest) : void
1.1: getUserId() : long: userId
1.2: createVote(userId : long, request : VoteRequest) : void
1.2.1: findById(id : Long) : Optional<User>: user
break
[user.empty()]
1.2.2: UnauthenticatedException
1.2.3: HTTP 401 Unauthorized
1.2.3: findById(id : long) : Optional<Comment>: comment
break
[comment.empty()]
1.2.4: CommentNotFoundException
1.2.4.1: HTTP 400 Bad Request
1.2.5: findByUser_IdAndComment_Id(userId : long, commentId : long) : Optional<Vote>
opt
[vote.empty()]
1.2.6:
vote : Vote
1.2.7: setVoteType(voteType : VoteType) : void
1.2.8: save(vote : Vote) : void

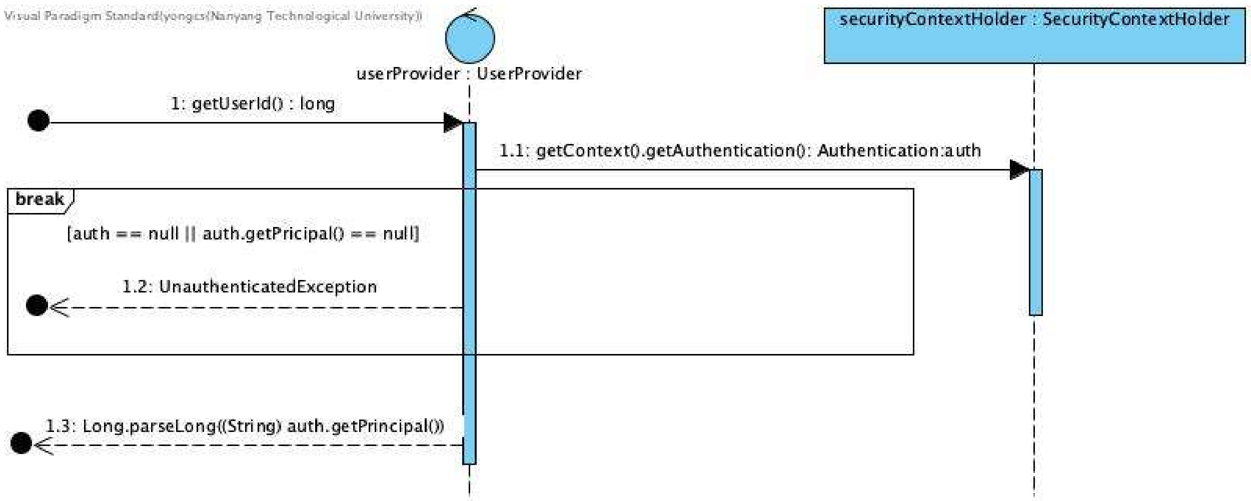
EB.11 SaveSchool


Visual Paradigm Standard(yongcs(Nanyang Technological University))
Client
userController : UserController
userProvider : UserProvider
userService : UserService
userRepository : UserRepository
dbSchoolRepository : DbSchoolRepository
user : User
1: saveSchool(request : SaveSchoolRequest) : void
1.1: getUserId() : long
1.2: saveSchool(userId : long, request : SaveSchoolRequest) : void
1.2.1: findById(id : Long) : Optional<User>: user
break
[user.empty()]
1.2.2: UnauthenticatedException
1.2.3: HTTP 401 Unauthorised
1.2.3: findById(id : long) : Optional<DbSchool>: school
break
[school.empty()]
1.2.4: SchoolNotFoundException
1.2.4.1: HTTP 400 Bad Request
opt
[request.isSaving()]
1.2.5: addSavedSchool(school : DbSchool) : void
1.2.6: deleteSavedSchool(school : DbSchool) : void
1.2.7: save(user : User) : void

EB.12 GetSavedSchools

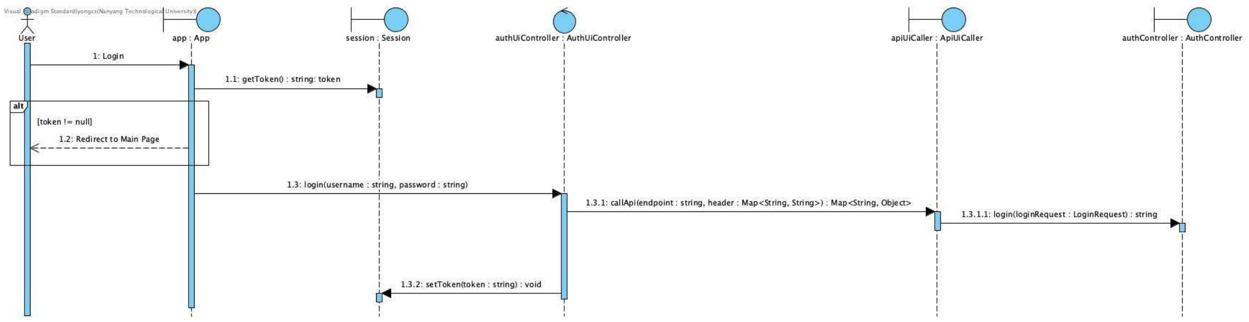

Visual Paradigm Standard(yongcs(Nanyang Technological University))
Client
userController : UserController
userProvider : UserProvider
userService : UserService
userRepository : UserRepository
user : User
dbSchool : DbSchool
1: getSavedSchoolIds() : List<Long>
1.1: getUserId() : long
1.2: getSavedSchoolIds(userId : long) : List<Long>
1.2.1: findById(id : Long) : Optional<User>
break
[user.empty()]
1.2.2: UnauthenticatedException
1.2.2.1: HTTP 401 Unauthorised
1.2.3: getSavedSchools() : List<DbSchool>: schools
loop
[DbSchool school:schools]
1.2.4: getId() : long

EB.13 GetUserContext

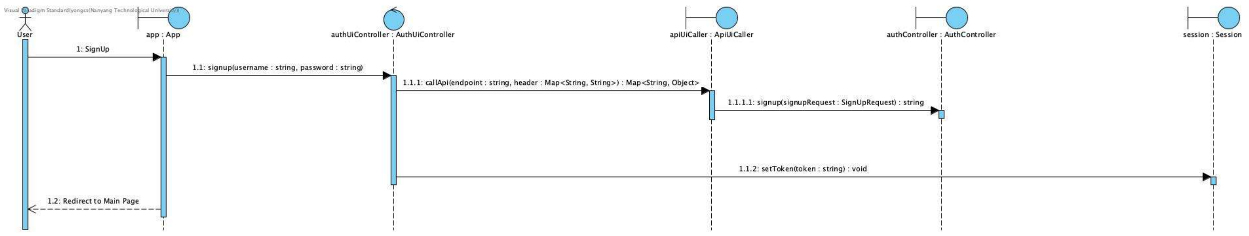

Visual Paradigm Standard(yongcs(Nanyang Technological University))
userProvider : UserProvider
securityContextHolder : SecurityContextHolder
1: getUserId() : long
1.1: getContext().getAuthentication(): Authentication:auth
break
[auth == null || auth.getPricipal() == null]
1.2: UnauthenticatedException
1.3: Long.parseLong((String) auth.getPrincipal())

Frontend

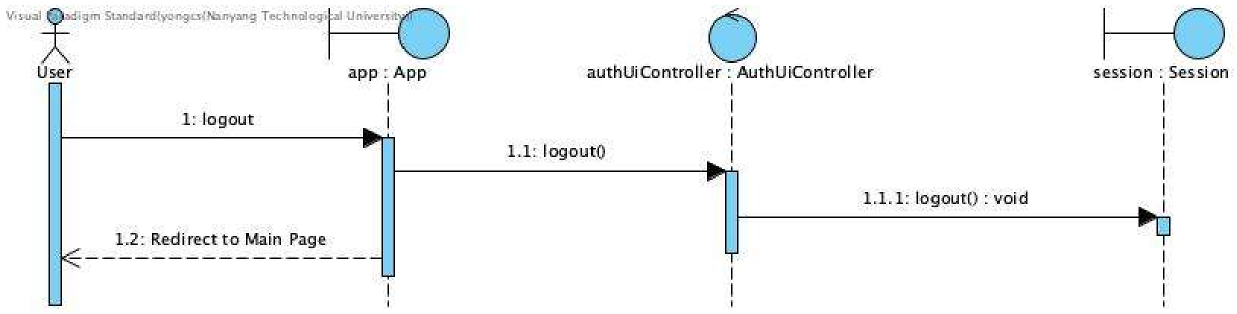
EF.1 Login


Visual Paradigm Standard(yongcs(Nanyang Technological University))
User
app : App
session : Session
authUiController : AuthUiController
apiUiCaller : ApiUiCaller
authController : AuthController
1: Login
1.1: getToken() : string: token
alt
[token != null]
1.2: Redirect to Main Page
1.3: login(username : string, password : string)
1.3.1: callApi(endpoint : string, header : Map<String, String>) : Map<String, Object>
1.3.1.1: login(loginRequest : LoginRequest) : string
1.3.2: setToken(token : string) : void

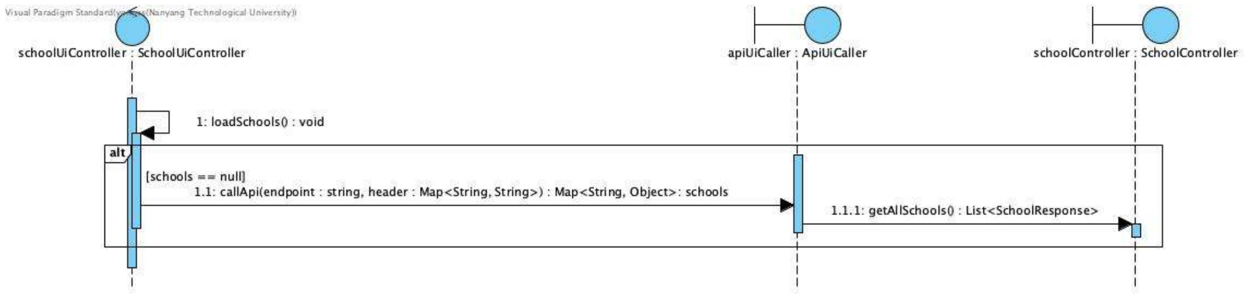
EF.2 Signup


Visual Paradigm Standard(yongcs(Nanyang Technological University))
User
app : App
authUiController : AuthUiController
apiUiCaller : ApiUiCaller
authController : AuthController
session : Session
1: SignUp
1.1: signup(username : string, password : string)
1.1.1: callApi(endpoint : string, header : Map<String, String>) : Map<String, Object>
1.1.1.1: signup(signupRequest : SignUpRequest) : string
1.1.2: setToken(token : string) : void
1.2: Redirect to Main Page

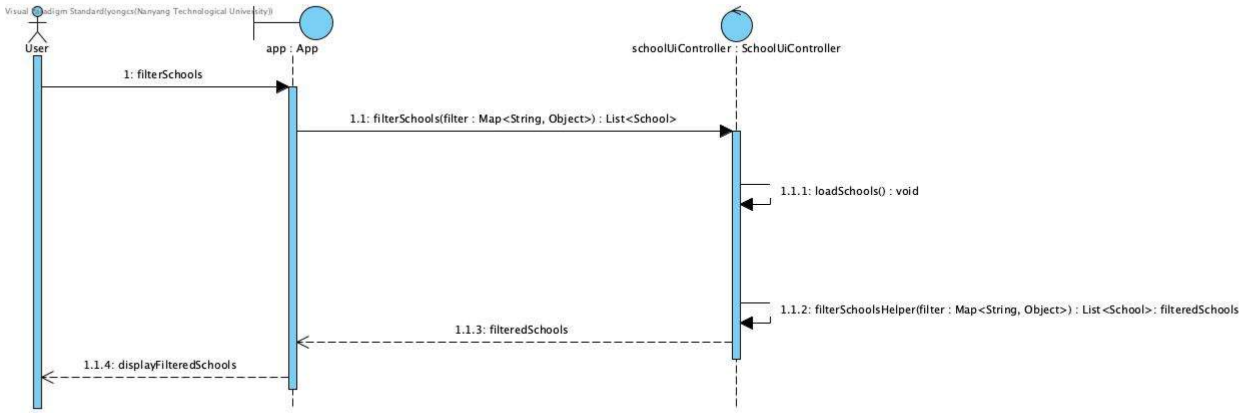
EF.3 Logout


Visual Paradigm Standard(yongcs(Nanyang Technological University))
User
app : App
authUiController : AuthUiController
session : Session
1: logout
1.1: logout()
1.1.1: logout() : void
1.2: Redirect to Main Page

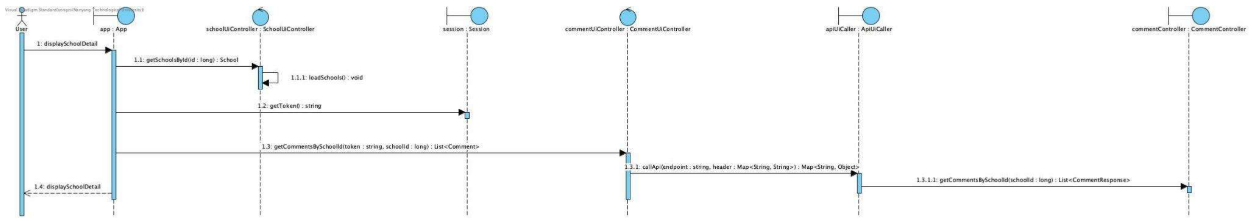
EF.4 LoadSchools


Visual Paradigm Standard(yongcs(Nanyang Technological University))
schoolUiController : SchoolUiController
apiUiCaller : ApiUiCaller
schoolController : SchoolController
1: loadSchools() : void
alt
[schools == null]
1.1: callApi(endpoint : string, header : Map<String, String>) : Map<String, Object>: schools
1.1.1: getAllSchools() : List<SchoolResponse>

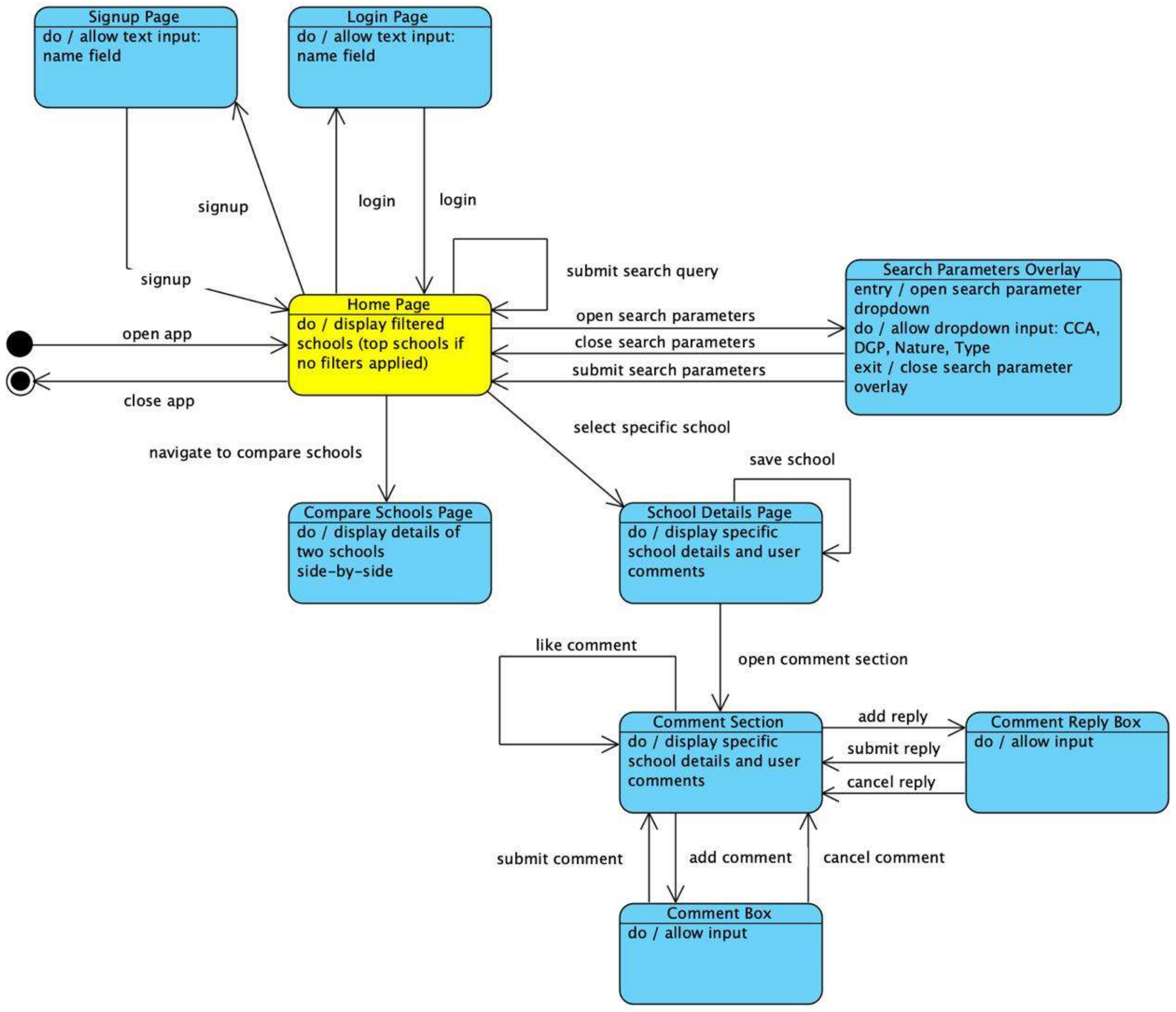
EF.5 FilterSchools


Visual Paradigm Standard(yongcs(Nanyang Technological University))
User
app : App
schoolUiController : SchoolUiController
1: filterSchools
1.1: filterSchools(filter : Map<String, Object>) : List<School>
1.1.1: loadSchools() : void
1.1.2: filterSchoolsHelper(filter : Map<String, Object>) : List<School>: filteredSchools
1.1.3: filteredSchools
1.1.4: displayFilteredSchools

EF.6 ViewSchoolDetails


Visual Paradigm Standard(yongcs(Nanyang Technological University))
User
app : App
schoolUiController : SchoolUiController
session : Session
commentUiController : CommentUiController
apiUiCaller : ApiUiCaller
commentController : CommentController
1: displaySchoolDetail
1.1: getSchoolsById(id : long) : School
1.1.1: loadSchools() : void
1.2: getToken() : string
1.3: getCommentsBySchoolId(token : string, schoolId : long) : List<Comment>
1.3.1: callApi(endpoint : string, header : Map<String, String>) : Map<String, Object>
1.3.1.1: getCommentsBySchoolId(schoolId : long) : List<CommentResponse>
1.4: displaySchoolDetail

F. Initial Dialog Map



Signup Page
do / allow text input: name field
Login Page
do / allow text input: name field
signup
login
login
signup
submit search query
Home Page
do / display filtered schools (top schools if no filters applied)
open search parameters
close search parameters
submit search parameters
Search Parameters Overlay
entry / open search parameter dropdown
do / allow dropdown input: CCA, DGP, Nature, Type
exit / close search parameter overlay
open app
close app
select specific school
navigate to compare schools
save school
Compare Schools Page
do / display details of two schools side-by-side
School Details Page
do / display specific school details and user comments
like comment
open comment section
Comment Section
do / display specific school details and user comments
add reply
submit reply
cancel reply
Comment Reply Box
do / allow input
submit comment
add comment
cancel comment
Comment Box
do / allow input