LinkedTek

T10 - SWA - Advanced Cloud

« Advanced Cloud »

Software Architecture Specifications

Table of contents

Figures List

Introduction

The aim of this software architecture specification (SAS) is to present the technical elements necessary for the **LikedTek** project.

1. Project context

The aim of this project is to provide a fully workable professional social network system, the main features will be the abilities to find and add new relations (thanks to some markers like schools, works or even relation of relation “friend of friend”), the abilities to publish some posts and comment them, to see on a single page all the relation’s posts (like a feed on Facebook, Twitter or LinkedIn) and to send messages to the other platform’s users.

2. Global architecture

The LikedTek project can easily be dived into two different parts, the frontend, and the backend part. The following part will describe the architecture of the frontend. The backend architecture specification can be found at the 4 part of this document.

3. Frontend architecture

For this project, the frontend part had been made using **JavaScript** language and more specifically, **Node.js**[1] and the **React**[2] framework.

Node.js is an asynchronous event driven JavaScript runtime. It is designed to build scalable network applications.

React is a library for building composable user interfaces. It encourages the creation of reusable UI components, which present data that changes over time.

In the Linkedtek project, the frontend uses as much as possible the **Material-UI**[3] framework to design the components.

Material-UI is an open-source project that features React components that implement Google's **Material Design**[4]**.**

A. General overview

While the frontend part is using React, it had been made with components. Those components are made on the same model.

At the top level, there is a folder called “Activities”. An activity can be defined by a page on the user interface.

The activity load component called “inner”. In this inner component, the grid of activity is defined.

The inner component loads the activity related sub-components and dispose them on the previously defined grid.

The communication between the frontend application and the backend gateway had been made using **Axios**[5] client. Axios is a promise-based HTTP client for the browser and node.js. More information about the used packages are available on the delivery procedure documentation.

You can find the activities and components description on the following part. This document will explain the component architecture and functionalities. The code functions are mentioned and explain, but only on the functional way. You can find more explanation about those functions into the code itself which is fully **JSDoc**[6] commented.

[1] https://nodejs.org/
[2] https://reactjs.org/
[3] https://material-ui.com/
[4] https://material.io /
[5] https://github.com/axios /
[6] https://devdocs.io/jsdoc/

B. Activities description

a. Login

The Login component is the first component loaded on the LikedTek React application. Like for the Register component, this component is not based on the model described above. Those two components are so simple that all the code had been made on the Activity part.

From this component, user can try to login or go to register page. The more important functions used in that component are so:

- *clickOnRegisterButton* function, that load and display the register page.
- *clickOnSubmitButton* function that try to connect user with previously filled credentials.

If login succeed, username and email are received as return of gateway. Those information are next stored into the browser local storage in order to be used in the application later.

b. Register

The Register component is made on the same model as the Login component. It can be load only from the Login component and allow user to create a new account on the LikedTek service. In order to register, new user have to fill some information:

- Email address
- Name
- Password
- Password confirmation.

If one or more of this field had not being filled, the registration failed, and user is pleased to fill missing field before to retry.

The unique function that is important is the *clickOnSubmitButton* function that handle the registration.

c. Feed

This component is made on the model described on the general overview part. The feed activity loads the menu component. The actuality inner component defines a single column grid. The actuality subcomponent is composed by the actuality feed who list the relation posts, ordered by date.

From this activity, user can:

- Click on any item of the left menu (more information on part j below).
- Click on disconnect icon to be disconnected from the service.
- Click on any post on feed list.

The most important functions of this component are:

- *componentWillMount* function. Like the name suggest, this function is called before the component mounting. In that function, the username and email are

loaded from the local storage. Then the actuality feed is asked to the backend gateway.
- *handleLoadActualityDetails* function. This function displays the post details activity. It also passes the post information to the next activity using the React props system. Those information are user email, post id, post title, post content, postdate, post owner.

This component is composed by files:

- *Actualities.js*
- *ActualitiesInner.js*
- *FeedModule.js*

d. Actuality details

This component is made on the model described on the general overview part. This component displays the details of post previously selected by user. For more information about the previous component, read the c part above.

The actuality details inner defines a 3 columns grid and load the required subcomponents :

- Post details is the subcomponent, that displays the post information.
- Comments list is the subcomponent, that displays the comments related to the selected post.
- Write new comment is the subcomponent, that allow user to add new comment to the post.

There is no action available for user on the post details submodule, it’s only displaying the post information.

There is no action available for user on the comment list subcomponent, it’s only the displaying the post related comments.

The most important functions of this component are:

- *componentWillMount* function. Like the post details are received from the parent component, the componentWillMount function handle only the request for the comments related to the selected post.
- *HandleAddNewComment* function. This function is called in order to add new comment to the selected post. There is a single text field allowing user to enter new comment and a button for validation. If the text field is empty, an error popup is displayed to warn user that the new comment creation failed.

This component is composed by files:

- *ActualitiesDetails.js*
- *ActualitiesDetailsInner.js*
- *AddCommentModule.js*

- *PostModule.js*
- *CommentModule.js*

e. Messages

This component is made on the model described on the general overview part. The messages inner defines a 3 columns grid and load the required subcomponents :

- Inbox subcomponent, that displays all the messages received by user.
- Outbox subcomponent, that displays all the messages send by user.
- Write new message subcomponent, that allow user to write new messages to his relations.

The inbox component allow user to reply to a received message.

There is no action available for user on the outbox subcomponent, it's only the displaying of the send messages.

The write new message subcomponent is composed by a drop-down list, to select the user relation recipient and two text fields, one for the message title, one for the message content.

The most important functions of this component are:

- *componentWillMount* function. In this function, component request the user relation list, the inbox and outbox list.
- *sendNewMessage* function. This function allow user to send new message to his relation. If the title or content text field is empty, an error popup is displayed to warn user that the message cannot be sent.
- *handleSendReplyMessage* function. This function allowing user the reply to a received message. This function differs from the sendNewMessage function in the way that it concatenates the received message content with the reply message content.

This component is composed by files:

- *Messages.js*
- *MessagesInner.js*
- *InboxModule.js*
- *OutboxModule.js*
- *WriteNewMessageModule.js*

f. Posts management

This component is made on the model described on the general overview part. The posts management inner defines a 3 columns grid and load the required subcomponents :

- Add new post and Relations posts subcomponent, that is used to publish a new post on the service and see the relation home page.
- My posts subcomponent, that displays the user post list.
- My comments subcomponents, that displays the user comments list.

On the add new post subcomponent, user can add publish a new post. There are two text fields, one for the post title and one for the post content. This subcomponent also adds a post button that validate the post. If one of the text fields is not filled, a popup is displayed to warn user that the post cannot be published.

On the relation posts, there is a single drop-down list where user select the relation profile and a button to validate. Here is also a popup displayed if user try to validate without any relation selected.

My posts subcomponent lists the user post, user can click on any post to display the post edit popup. From this popup, user can edit post content or title, view post details with all comments or delete the selected post.

My comments subcomponent lists the user comments related to relation post. User can click on any comment to display the comment edit popup. From this popup, user can edit or delete the selected comment.

The most important functions of this component are:

- *componentWillMount* function. In this component, the componentWillMount function will request the posts list, the user comments list and the user relation list.
- *handleRemovePost* function. This function is called when user want to delete a post.
- *handleEditPost function.* This function is called when user edit a selected post.
- *handleEditComment* function. This function is called when user edit a selected comment.
- *handleRemoveComment* function. This function is called when user want to delete a comment.
- *handleNewPost* function. This function is called when user publish a post.
- *loadActualityDetails* function. This function is called when user want to see a post details.
- *handleUserProfile* function. This function is called when user view a selected relation profile.

This component is composed by files:

- *PostManagement.js*
- *PostManagementInner.js*
- *AddPostModule.js*
- *MyPostsModule.js*
- *MyCommentsModule.js*

g. Profile

This component is made on the model described on the general overview part. The profile inner defines a 3 columns grid and load the required subcomponents :

- User profile module, that displays the current profile of user
- Job timeline module, that displays the current user job and the user job timeline.
- View user page, that allows user to see relations profiles.

For this activity, user can:

- Edit his own profile
- Add item to job timeline
- Edit item into job timeline
- View another user profile page

The profile subcomponent is composed by the user profile picture and a user profile description. User can click on the edit profile button to open the edit profile modal popup.

The timeline subcomponent is composed by the current user job, the user timeline list and a button to add new input to the timeline. If user click on it, it's opened the add new item modal popup. If user click on any item into the timeline, it's opened the edit timeline input modal.

The view user subcomponent is composed by a text field to enter username and a button to search user. If user click on the validate button with an empty text field, an alert is displayed to warn user that the search cannot be done. When search succeeds, the search result list is displayed. If user click on any item into the list, the view user profile modal popup is displayed.

The most important functions of this component are:

- *componentWillMount* function. For this component, it will request the user profile details, the country list, the company list and the user job list.
- *handleProfileModalClosevalidated* function. This function is called for updating the user profile.
- *handleUserModalCloseValidated* function. This function loads and displays the selected user profile.
- *handleJobInputModalCloseValidated* function. This function adds new job input into the timeline.
- *searchUserByName* function. This function is called when user search for any relation profile.

This component is composed by files:

- *Profile.js*
- *ProfileInner.js*
- *profileModule.js*
- *searchModule.js*
- *profileCurrentDescription.js*
- *timelineModule.js*
- *timelineList.js*

h. Relations

This component is made on the model described on the general overview part. The relations inner defines a 3 columns grid and load the required subcomponents :

- Add new relation subcomponent, that allow user to add user relation.
- Your relationships subcomponent, that displays the user relationships.
- Relation suggestions subcomponent that displays a suggestion list of users, schools and companies, based on the user current relations.

From this activity, user can:

- Search user by his name
- Add user to his relation
- Display selected user profile

Add new relation subcomponent is compose by a single text field for the search user by name and a button to validate. If the text field is empty when user click on the search button, a popup is displayed to warn user that the searching user failed.

Your relationships subcomponent displays the user current relations. User can click on any relation to displays the relation information popup. From that popup, user can only see the selected user profile page.

Relation suggestions subcomponent is used to show user potential relations interests. If user click on any relation on the list, it displays the relation information popup. From that popup, user can only add selected user as relation.

The most important functions of this component are:

- *componentWillMount* function. In this component, this function will request for the user relation list and the user relation suggestions.
- *handleRemoveRelation* function. This function is used to remove the selected relation from user relations.
- *handleRelationModalCloseValidated* function. This function is called when user click on view profile of a selected relation.
- *searchUserByName* function. This function is called when user search any user.

This component is composed by files:

- *Relations.js*
- *RelationsInner.js*
- *AddRelationModule.js*
- *DisplayRelationModule.js*
- *RelationSuggestionModule.js*

i. Schools and companies

This component is made on the model described on the general overview part. The schools & companies inner defines a 3 columns grid and load the required subcomponents :

- Add new input subcomponent, that allow user to add new school or company to the LinkedTek service.
- School list subcomponent, that displays the list of schools available on the service.
- Companies list subcomponent, that displays the list of companies available in the service.

From this activity, user can :

- Add new input to school or company list
- Subscribe or unsubscribe to company or school
- Edit school or company name
- Edit school or company description

Add new input subcomponent is composed by a checkbox list to select the input type (school or company), two text fields for input name and description, a drop-down list to select the new input country and a validate button. If user click on validate button with one of the item listed above not filled, a popup is displayed to warn user that the new input creation failed.

The schools list component is composed by a drop-down list, allowing user to filter the schools by country, a list of schools with on each row, a button which allow user to subscribe or unsubscribe to school relation, depending on user relation state with this school.

The companies list component is composed by a drop-down list, allowing user to filter the companies by country, a list of companies with on each row, a button which allow user to subscribe or unsubscribe to company relation, depending on user relation state with this company.

The most important functions of this component are:

- *componentWillMount* function. For this component, the function will request, the countries list, the schools list, the companies list, user school subscriptions list and company subscriptions list.
- *addNewSchoolOrCompany* function. This function is called to create a new school or a new company.
- *handleSchoolOrCompanySubscription function. This function is used to subscribe or unsubscribe user to school or company.*
- *handleEditCompanyModalCloseValidated* function. This function is used to edit a company description and name.
- *handleEditSchoolModalCloseValidated* function. This function is used to edit a school description and name.
- *handleFilterCountryChangeSchool* function. This function is used to filter schools by country.
- *handleFilterCountryChangeCompany* function. This function is used to filter companies by country.

This component is composed by files:

- *SchoolsAndCompanies.js*
- *SchoolsAndCompaniesInner.js*
- *AddNewSchoolOrCompany.js*
- *MySubscriptionSchoolsModule.js*
- *MySubscriptionCompaniesModule.js*

j. Left menu

The left menu is a component that differs from the previously described components. Like the menu is available from all pages (except for login and register), it’s loaded from each activity component.

The component is composed by a single list. Each row in the list are composed by a listItem button, an icon and a text presentation.

If user clicks on any item in the list, it displays the related page.

From this component, user can:

- Load actualities feed page
- Load schools and companies page
- Load profile page
- Load post management page
- Load Relation page
- Load messages page

The functions available on this component are the onClick function of list item row. Those function used the ReactDom to display component on the screen.

C. Frontend activity diagrams

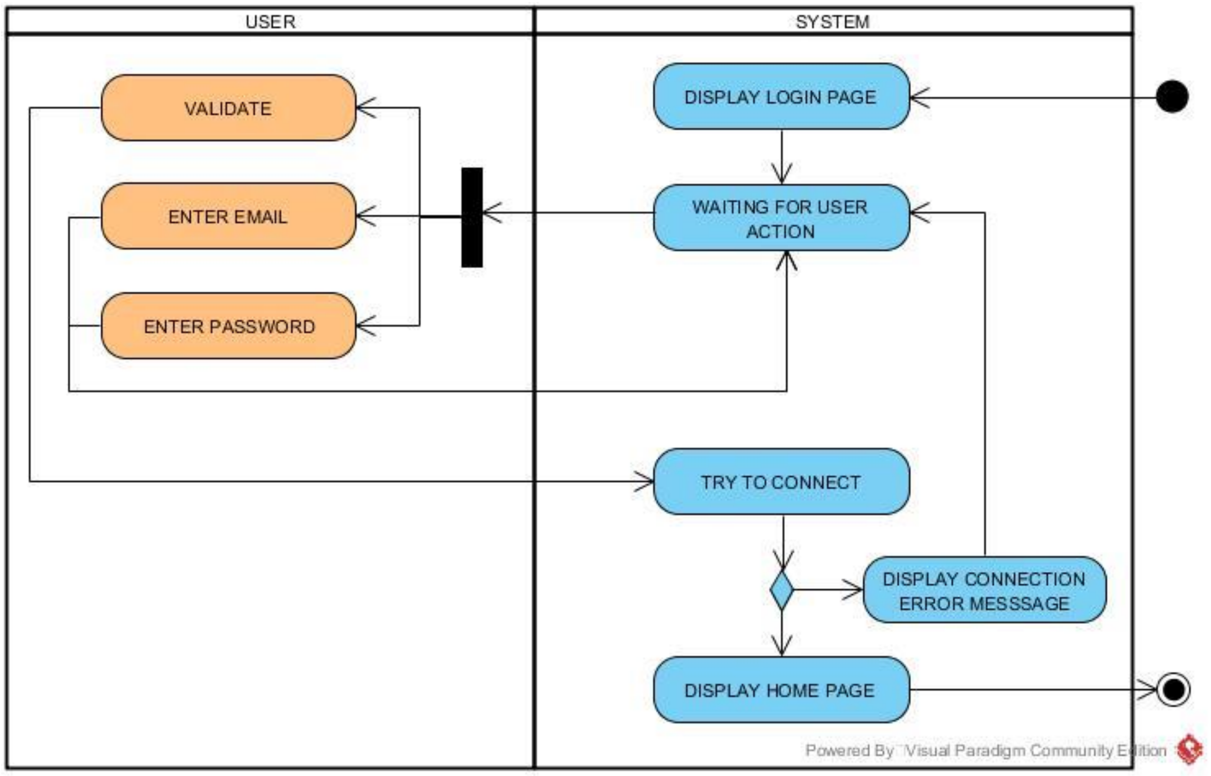
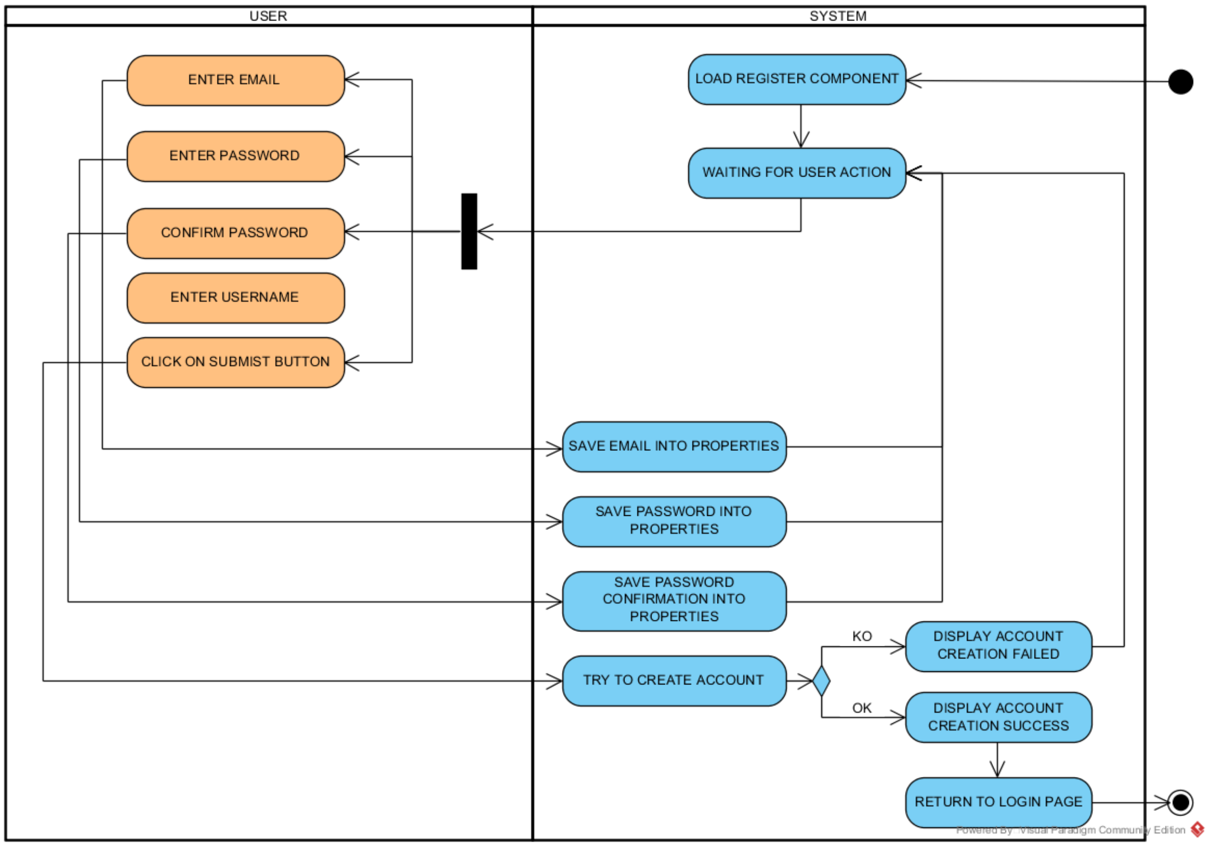


Figure 3-1 : Frontend login activity diagram



USER
SYSTEM
ENTER EMAIL
ENTER PASSWORD
CONFIRM PASSWORD
ENTER USERNAME
CLICK ON SUBMIT BUTTON
LOAD REGISTER COMPONENT
WAITING FOR USER ACTION
SAVE EMAIL INTO PROPERTIES
SAVE PASSWORD INTO PROPERTIES
SAVE PASSWORD CONFIRMATION INTO PROPERTIES
TRY TO CREATE ACCOUNT
KO
OK
DISPLAY ACCOUNT CREATION FAILED
DISPLAY ACCOUNT CREATION SUCCESS
RETURN TO LOGIN PAGE
Powered By Visual Paradigm Community Edition

Figure 3-2 : Frontend register activity diagram

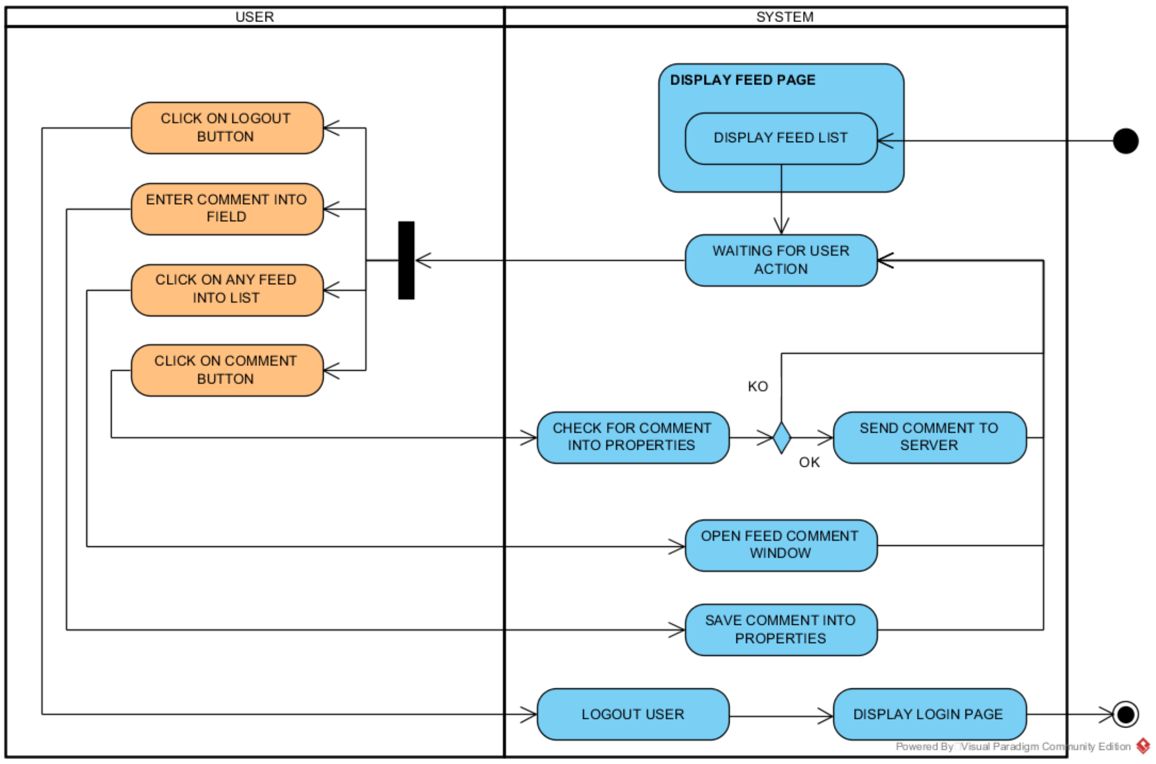


Figure 3-3 : Frontend actuality feed activity diagram

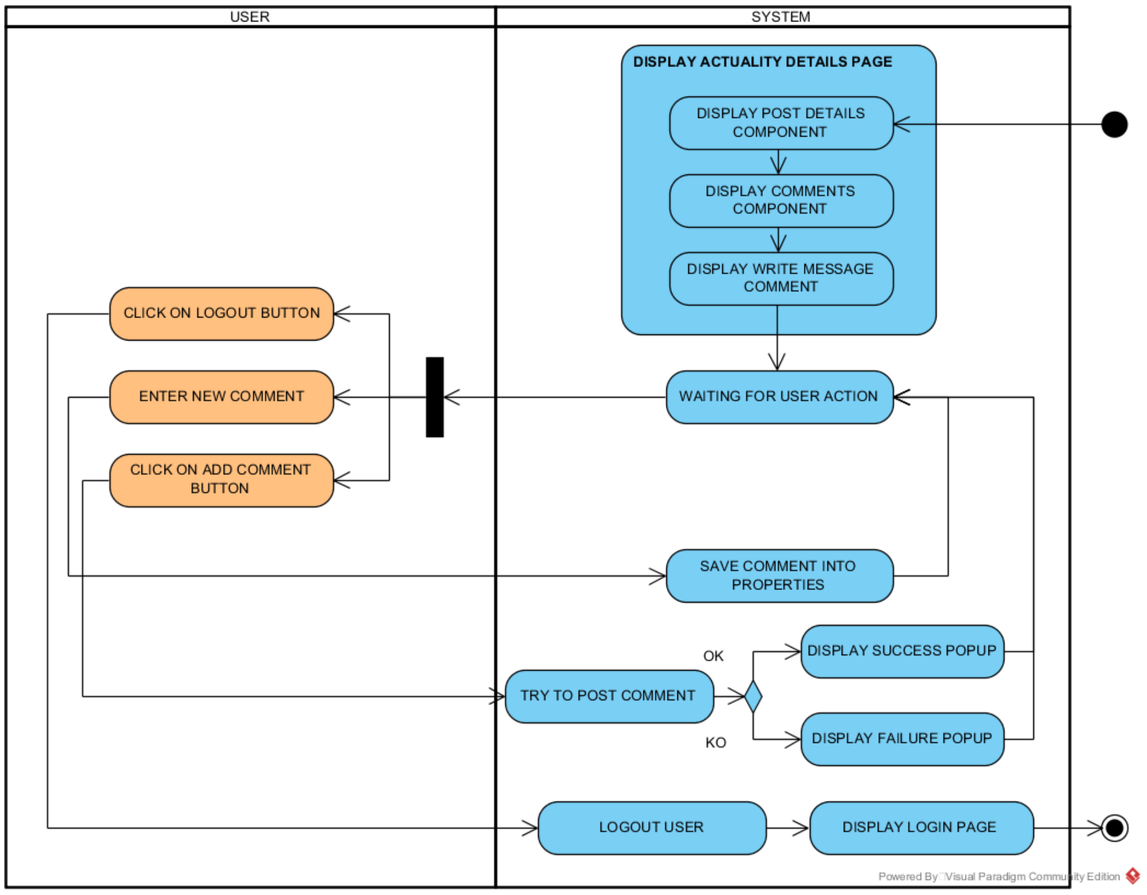
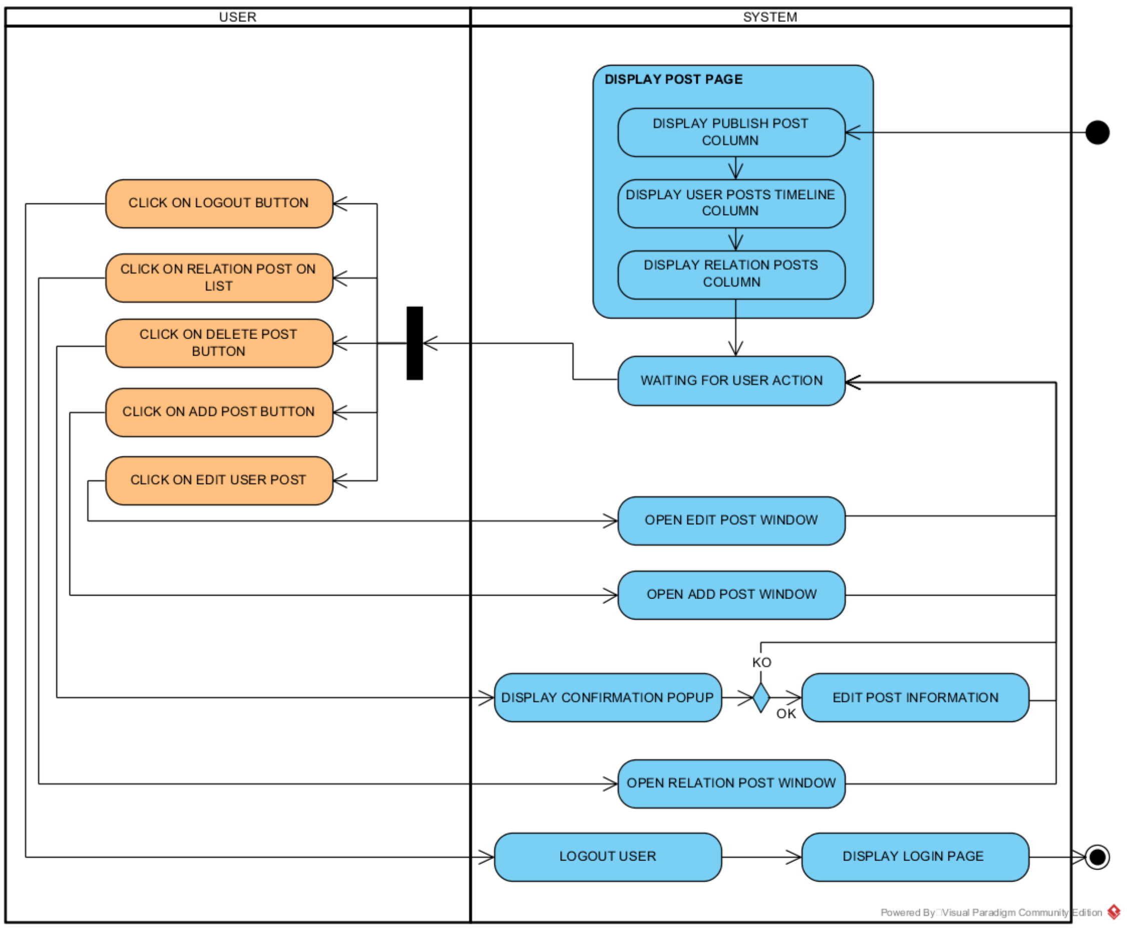

USER
SYSTEM
DISPLAY ACTUALITY DETAILS PAGE
DISPLAY POST DETAILS COMPONENT
DISPLAY COMMENTS COMPONENT
DISPLAY WRITE MESSAGE COMMENT
CLICK ON LOGOUT BUTTON
ENTER NEW COMMENT
CLICK ON ADD COMMENT BUTTON
WAITING FOR USER ACTION
SAVE COMMENT INTO PROPERTIES
OK
DISPLAY SUCCESS POPUP
TRY TO POST COMMENT
KO
DISPLAY FAILURE POPUP
LOGOUT USER
DISPLAY LOGIN PAGE
Powered By Visual Paradigm Community Edition

Figure 3-4 : Frontend actuality details activity diagram



USER
SYSTEM
DISPLAY POST PAGE
DISPLAY PUBLISH POST COLUMN
DISPLAY USER POSTS TIMELINE COLUMN
DISPLAY RELATION POSTS COLUMN
CLICK ON LOGOUT BUTTON
CLICK ON RELATION POST ON LIST
CLICK ON DELETE POST BUTTON
CLICK ON ADD POST BUTTON
CLICK ON EDIT USER POST
WAITING FOR USER ACTION
OPEN EDIT POST WINDOW
OPEN ADD POST WINDOW
KO
DISPLAY CONFIRMATION POPUP
OK
EDIT POST INFORMATION
OPEN RELATION POST WINDOW
LOGOUT USER
DISPLAY LOGIN PAGE
Powered By Visual Paradigm Community Edition

Figure 3-5 : Frontend post management activity diagram

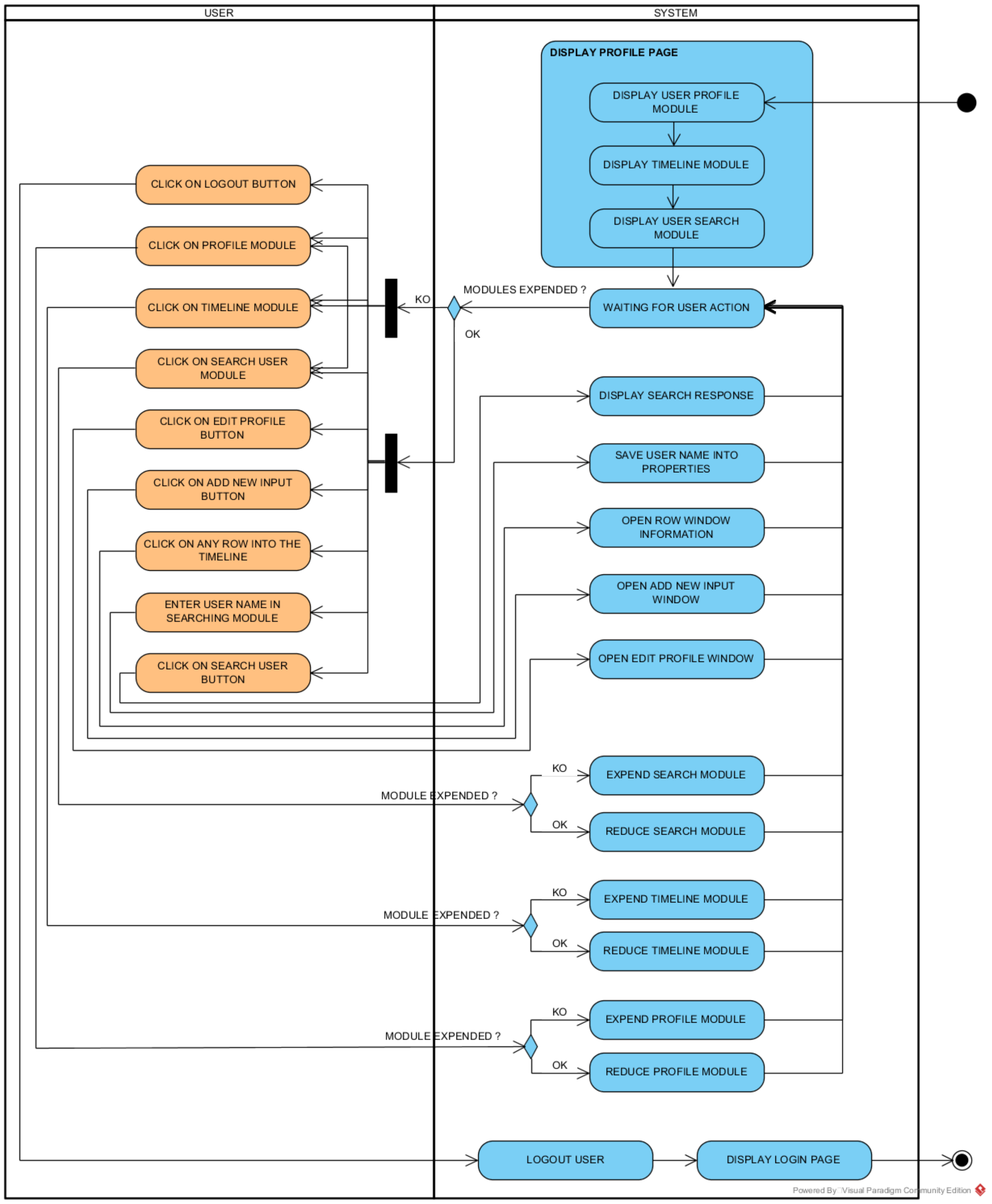


Figure 3-6 : Frontend profile activity diagram

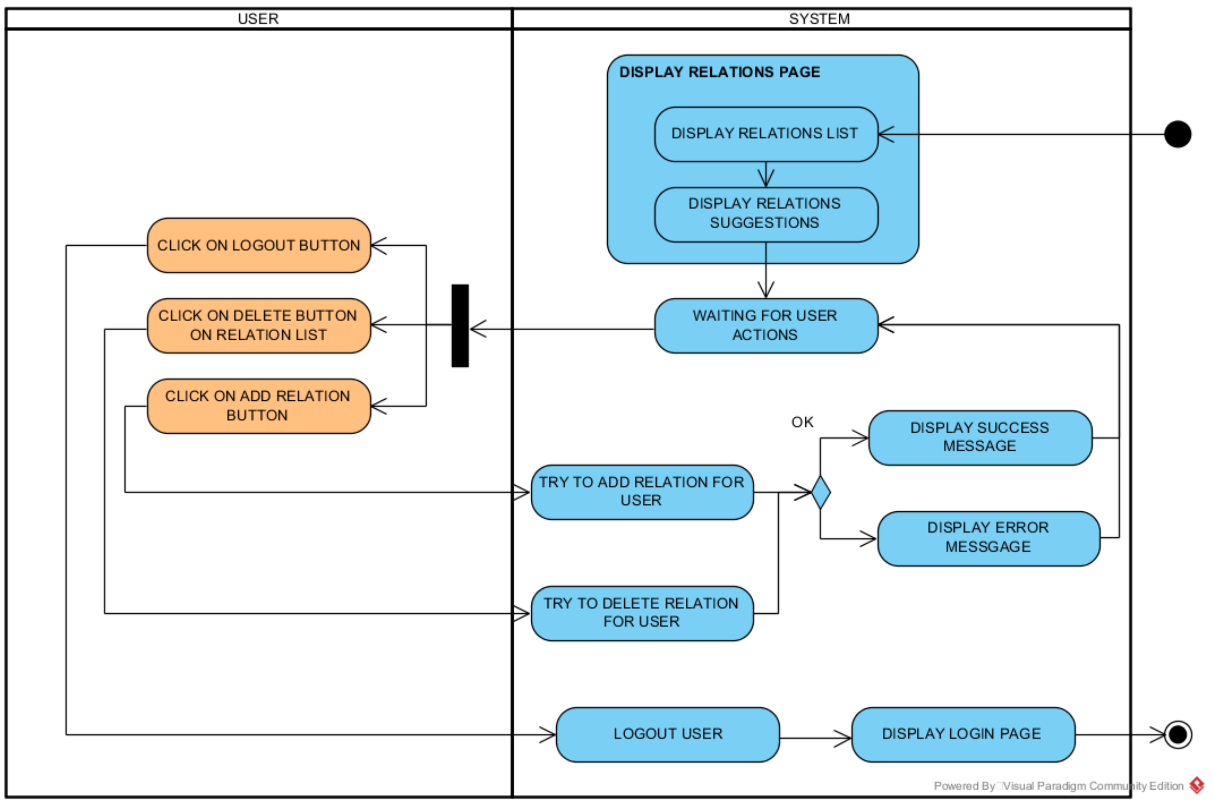

USER
SYSTEM
DISPLAY RELATIONS PAGE
DISPLAY RELATIONS LIST
DISPLAY RELATIONS SUGGESTIONS
CLICK ON LOGOUT BUTTON
CLICK ON DELETE BUTTON ON RELATION LIST
CLICK ON ADD RELATION BUTTON
WAITING FOR USER ACTIONS
OK
DISPLAY SUCCESS MESSAGE
TRY TO ADD RELATION FOR USER
DISPLAY ERROR MESSGAGE
TRY TO DELETE RELATION FOR USER
LOGOUT USER
DISPLAY LOGIN PAGE
Powered By Visual Paradigm Community Edition

Figure 3-7 : Frontend relations activity diagram

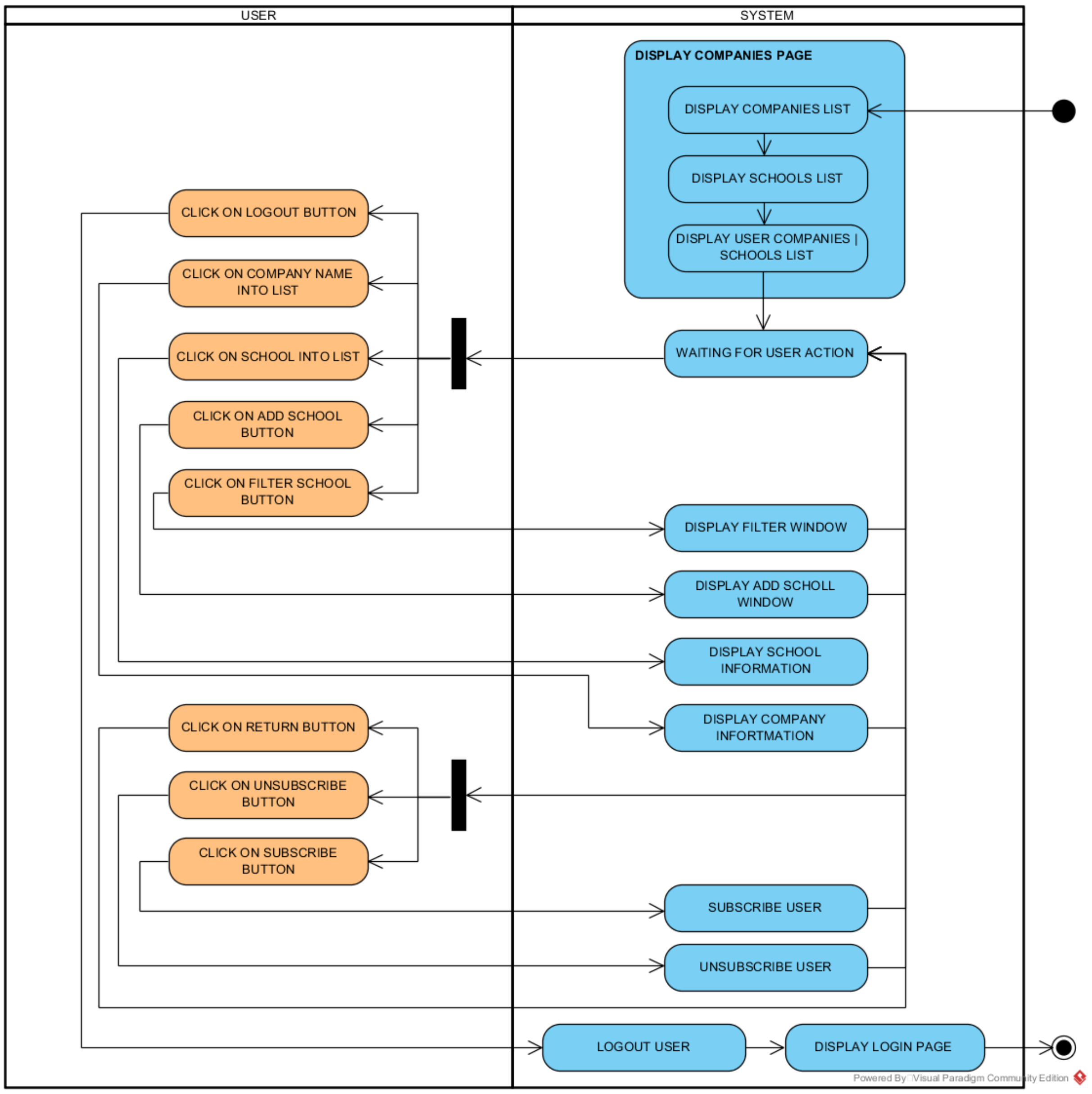

USER
SYSTEM
DISPLAY COMPANIES PAGE
DISPLAY COMPANIES LIST
DISPLAY SCHOOLS LIST
DISPLAY USER COMPANIES | SCHOOLS LIST
WAITING FOR USER ACTION
CLICK ON LOGOUT BUTTON
CLICK ON COMPANY NAME INTO LIST
CLICK ON SCHOOL INTO LIST
CLICK ON ADD SCHOOL BUTTON
CLICK ON FILTER SCHOOL BUTTON
DISPLAY FILTER WINDOW
DISPLAY ADD SCHOLL WINDOW
DISPLAY SCHOOL INFORMATION
DISPLAY COMPANY INFORTMATION
CLICK ON RETURN BUTTON
CLICK ON UNSUBSCRIBE BUTTON
CLICK ON SUBSCRIBE BUTTON
SUBSCRIBE USER
UNSUBSCRIBE USER
LOGOUT USER
DISPLAY LOGIN PAGE
Powered By Visual Paradigm Community Edition

Figure 3-8 : Frontend school and company activity diagram

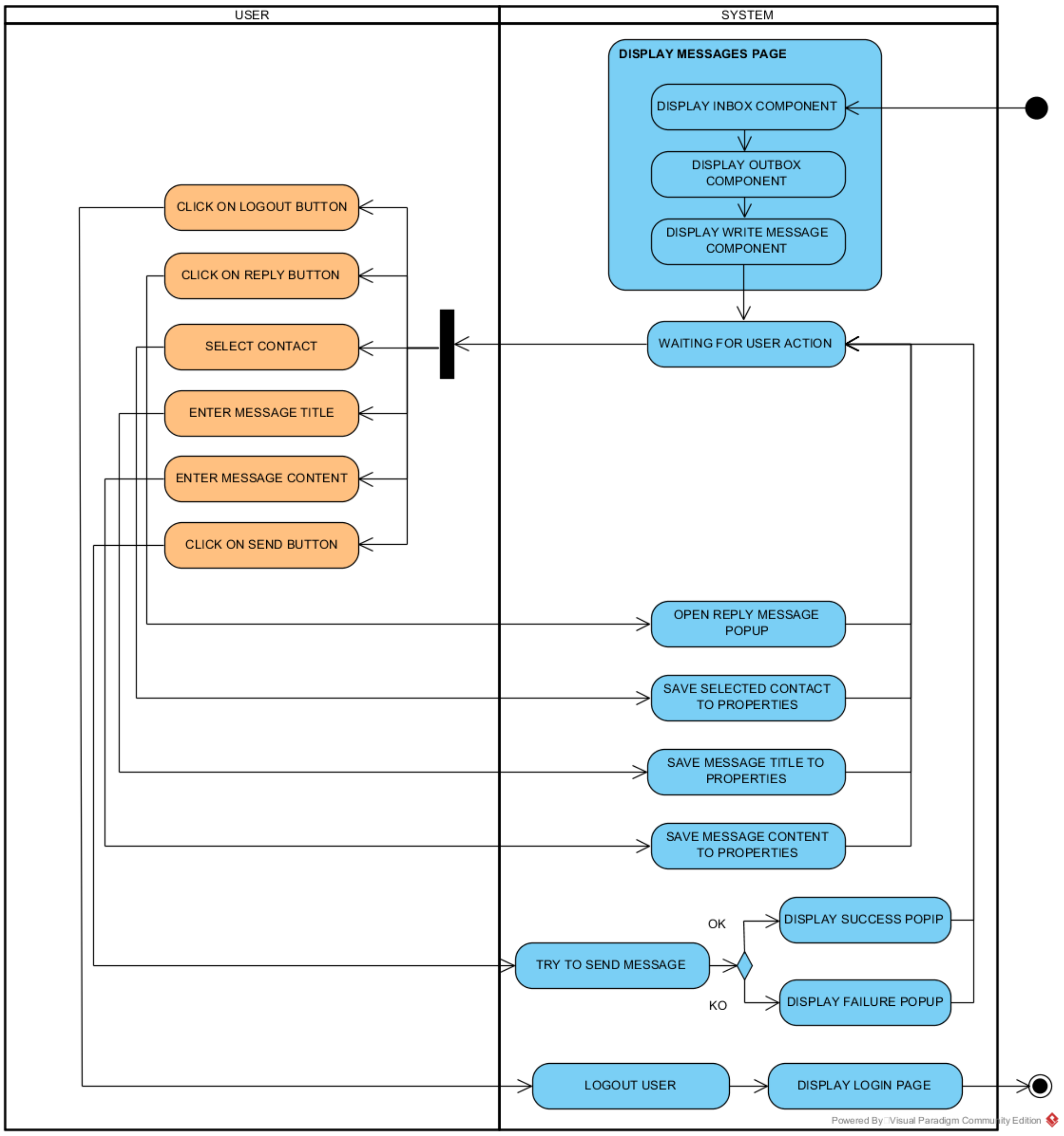


Figure 3-9 : Frontend messages activity diagram

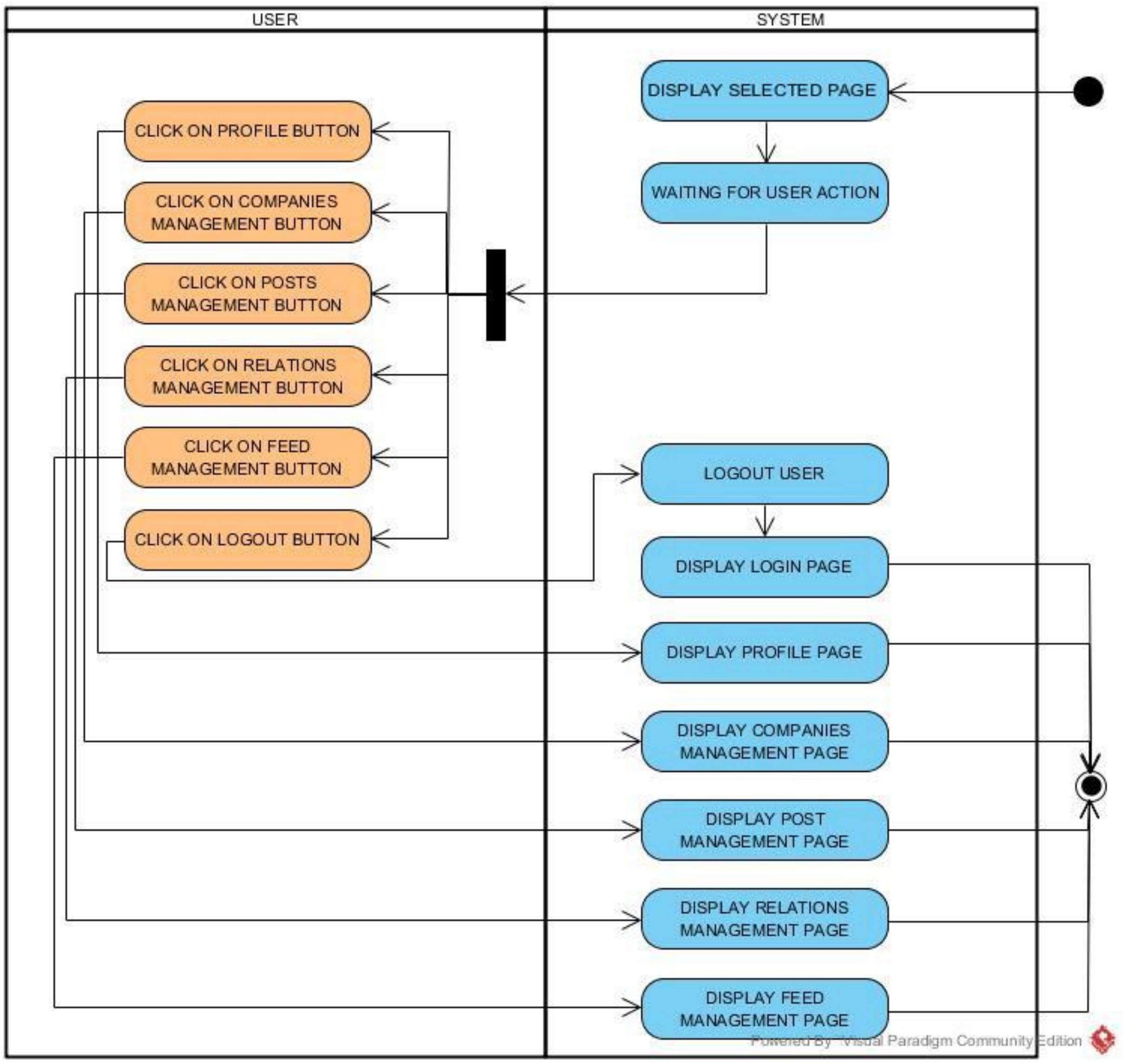

USER
SYSTEM
DISPLAY SELECTED PAGE
WAITING FOR USER ACTION
CLICK ON PROFILE BUTTON
CLICK ON COMPANIES MANAGEMENT BUTTON
CLICK ON POSTS MANAGEMENT BUTTON
CLICK ON RELATIONS MANAGEMENT BUTTON
CLICK ON FEED MANAGEMENT BUTTON
CLICK ON LOGOUT BUTTON
LOGOUT USER
DISPLAY LOGIN PAGE
DISPLAY PROFILE PAGE
DISPLAY COMPANIES MANAGEMENT PAGE
DISPLAY POST MANAGEMENT PAGE
DISPLAY RELATIONS MANAGEMENT PAGE
DISPLAY FEED MANAGEMENT PAGE
Powered By Visual Paradigm Community Edition

Figure 3-10 : Frontend menu activity diagram

D. Frontend mock-ups design

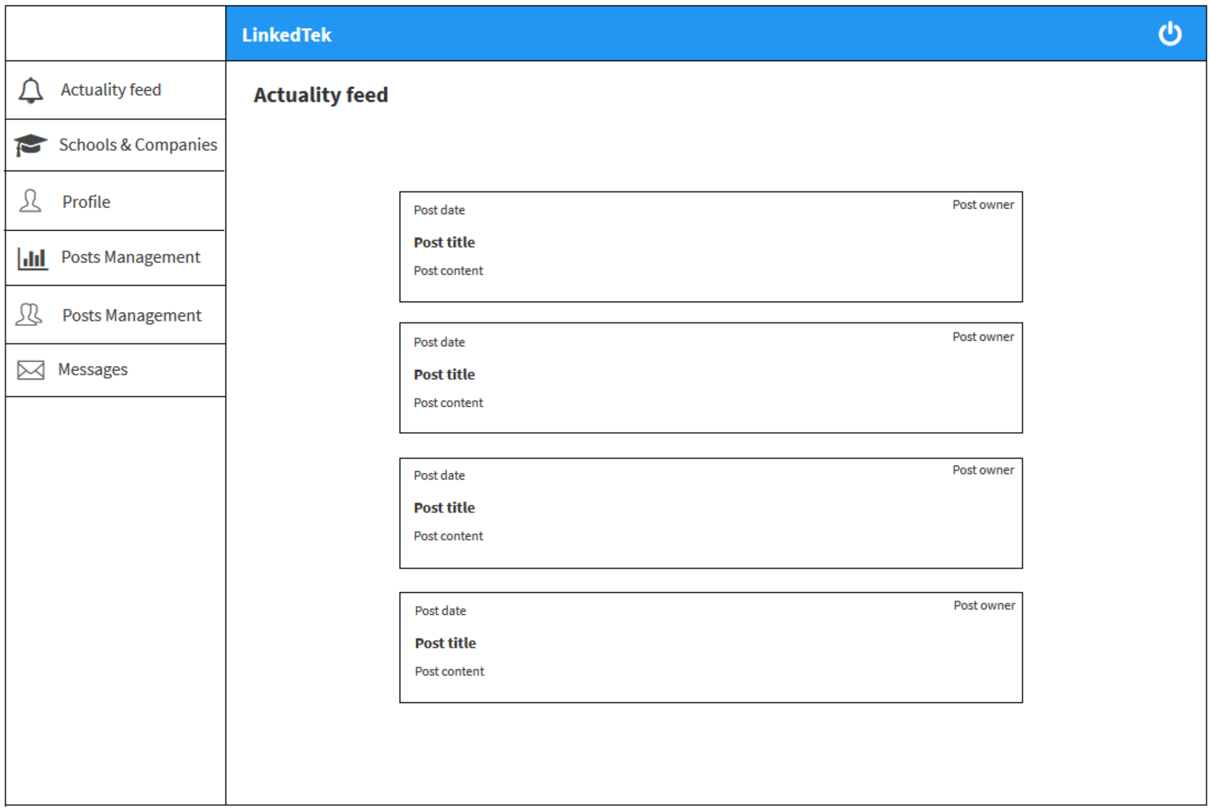


Figure 3-11 : Frontend actuality feed mockup

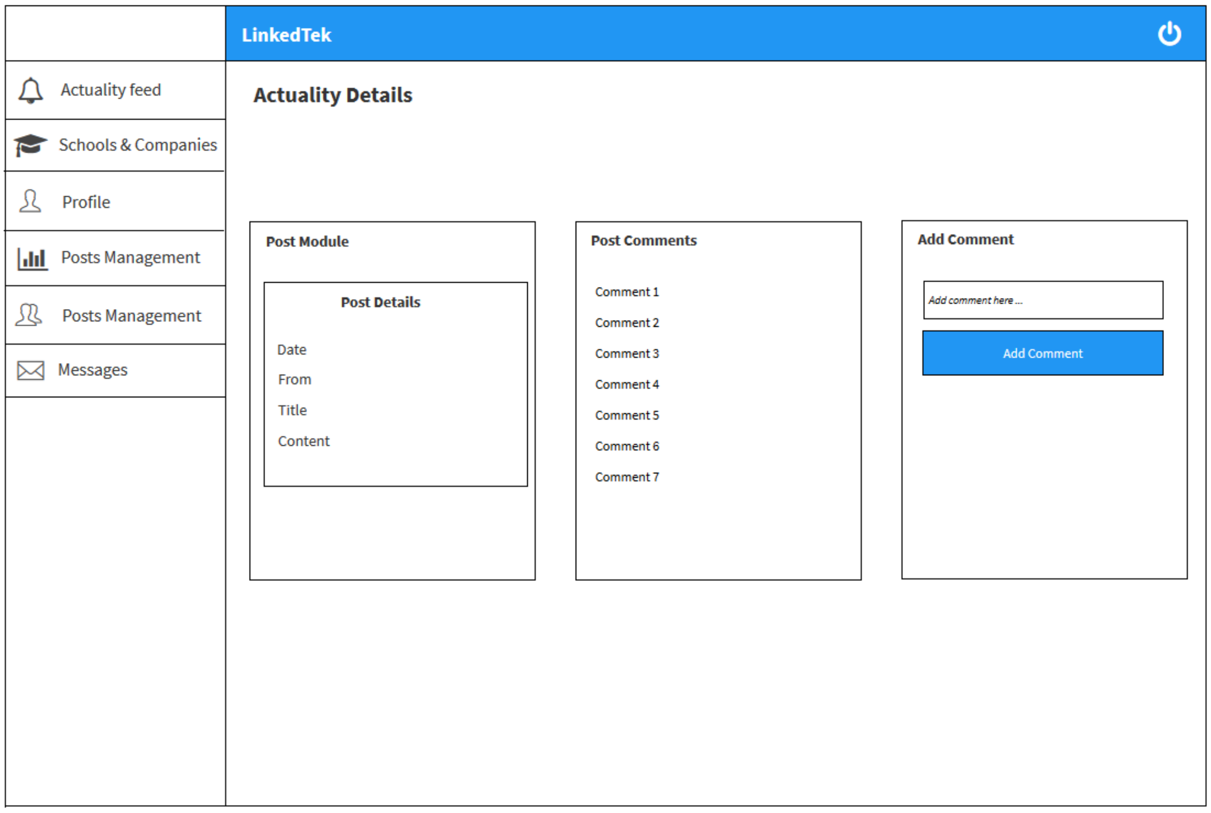


Figure 3-12 : Frontend actuality details mockup

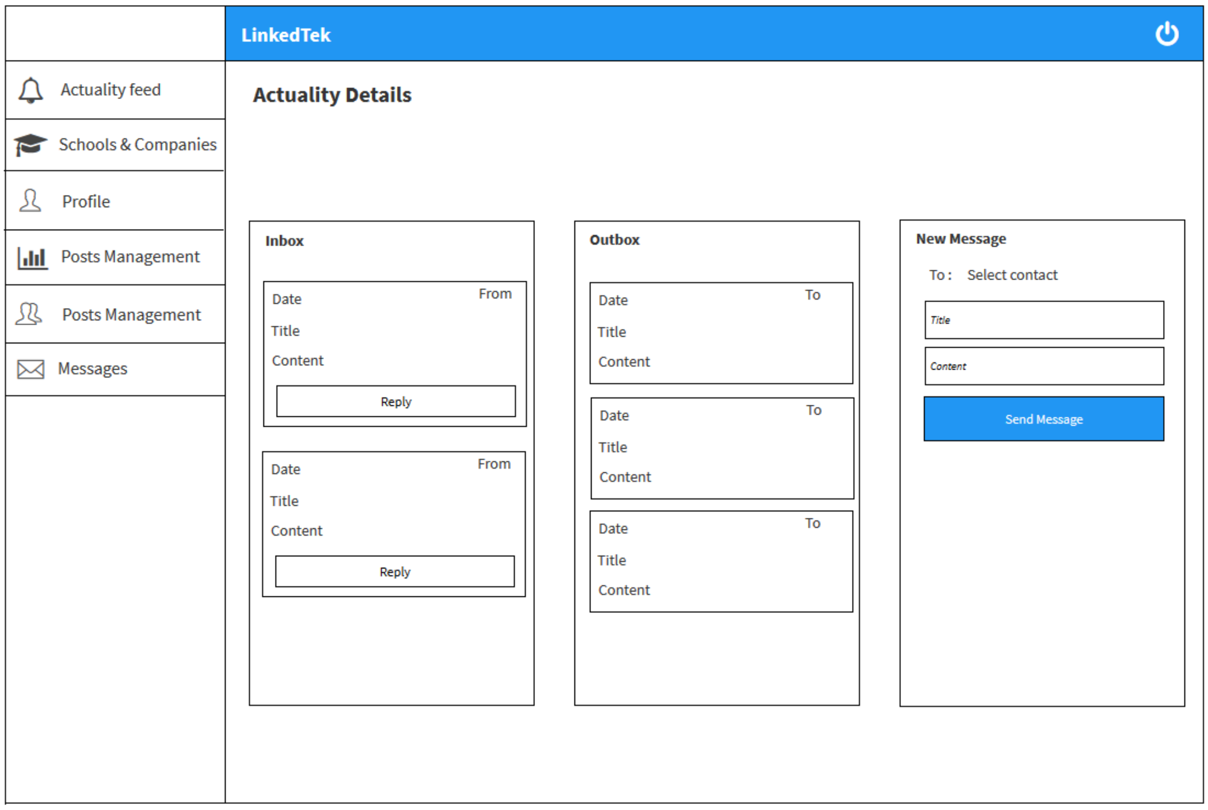


Figure 3-13 : Frontend messages mockup

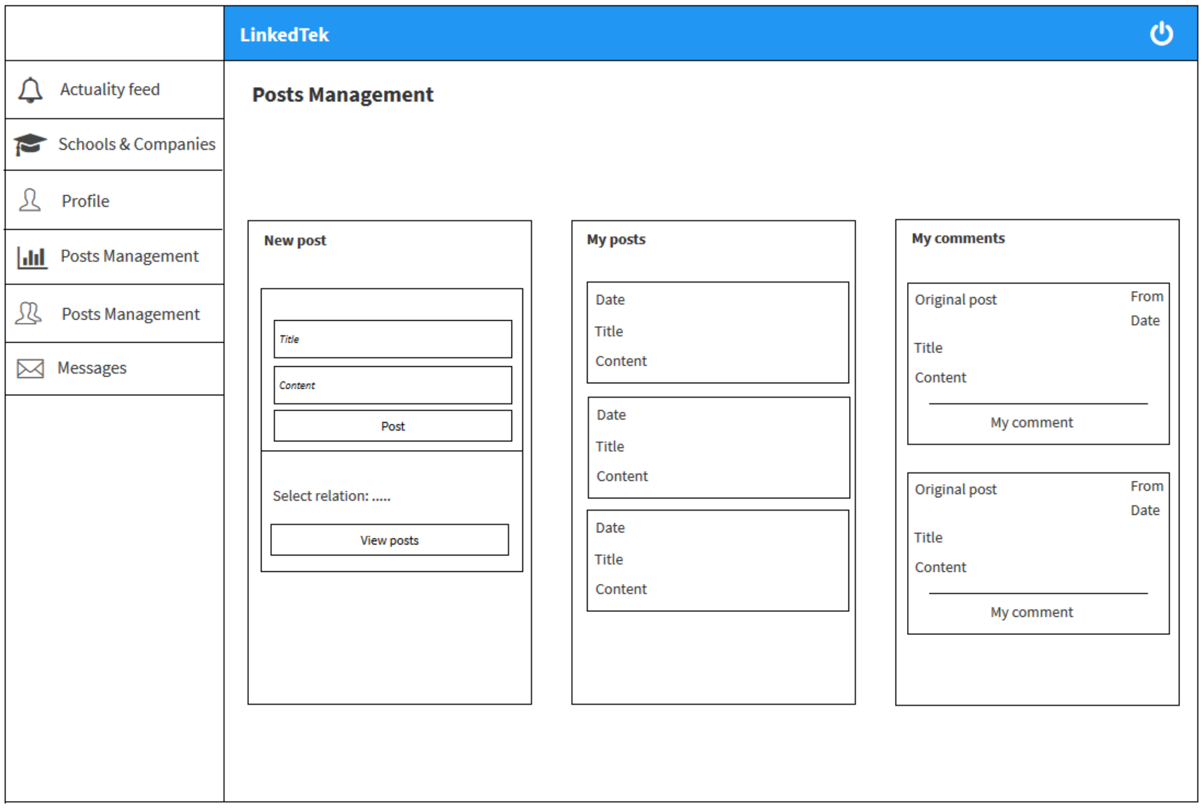


Figure 3-14 : Frontend posts management mockup

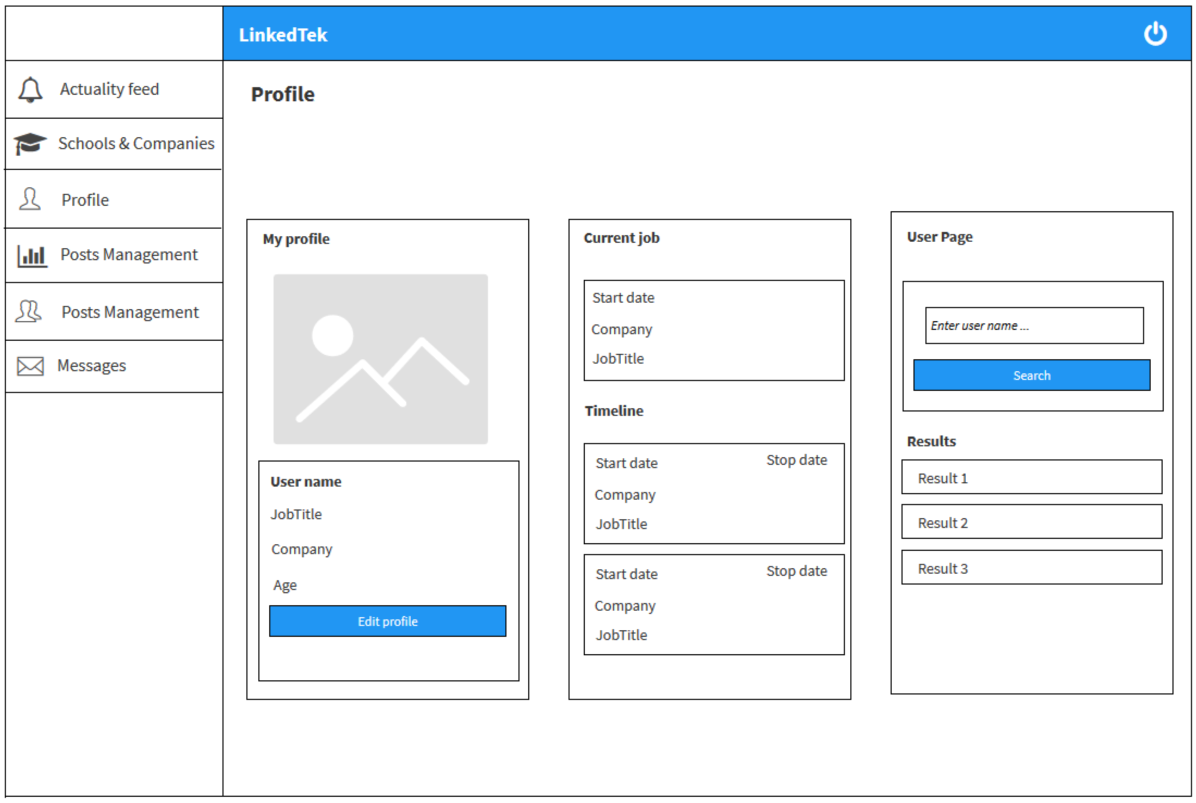


Figure 3-15 : Frontend user profile mockup

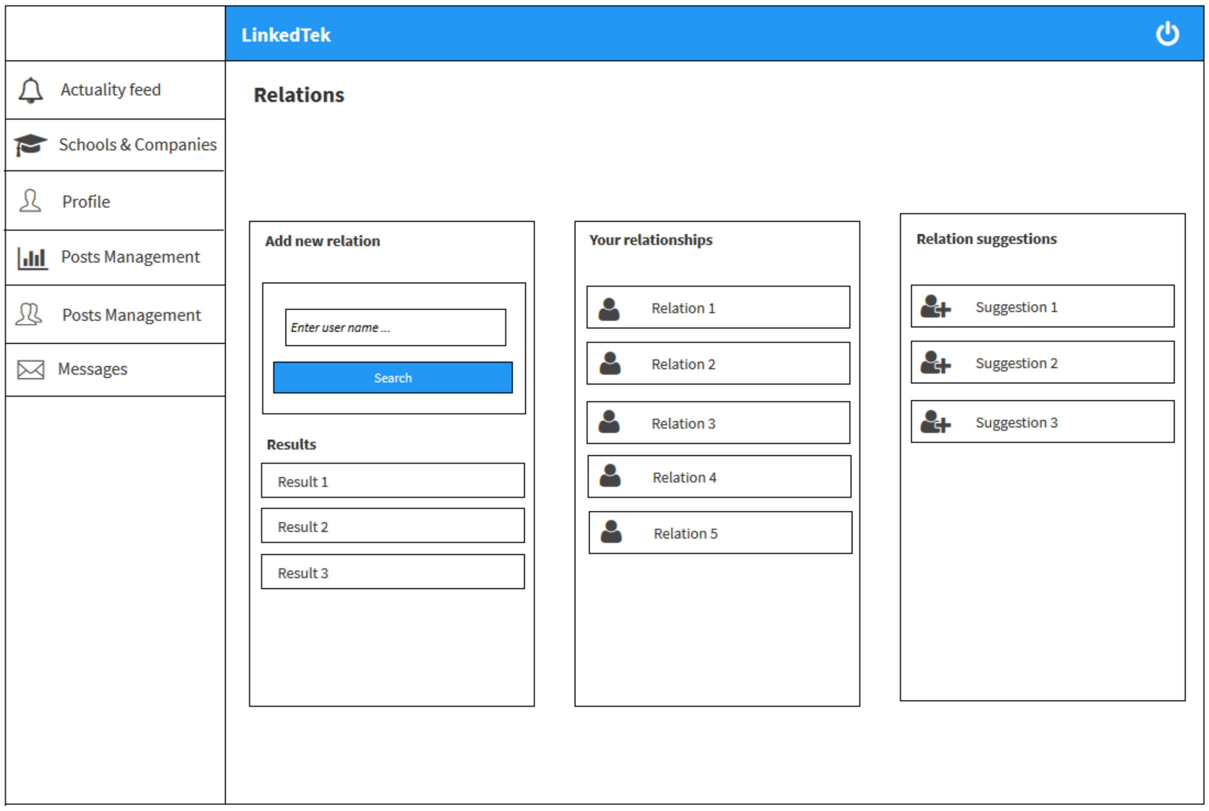


Figure 3-16 : Frontend relations mockup

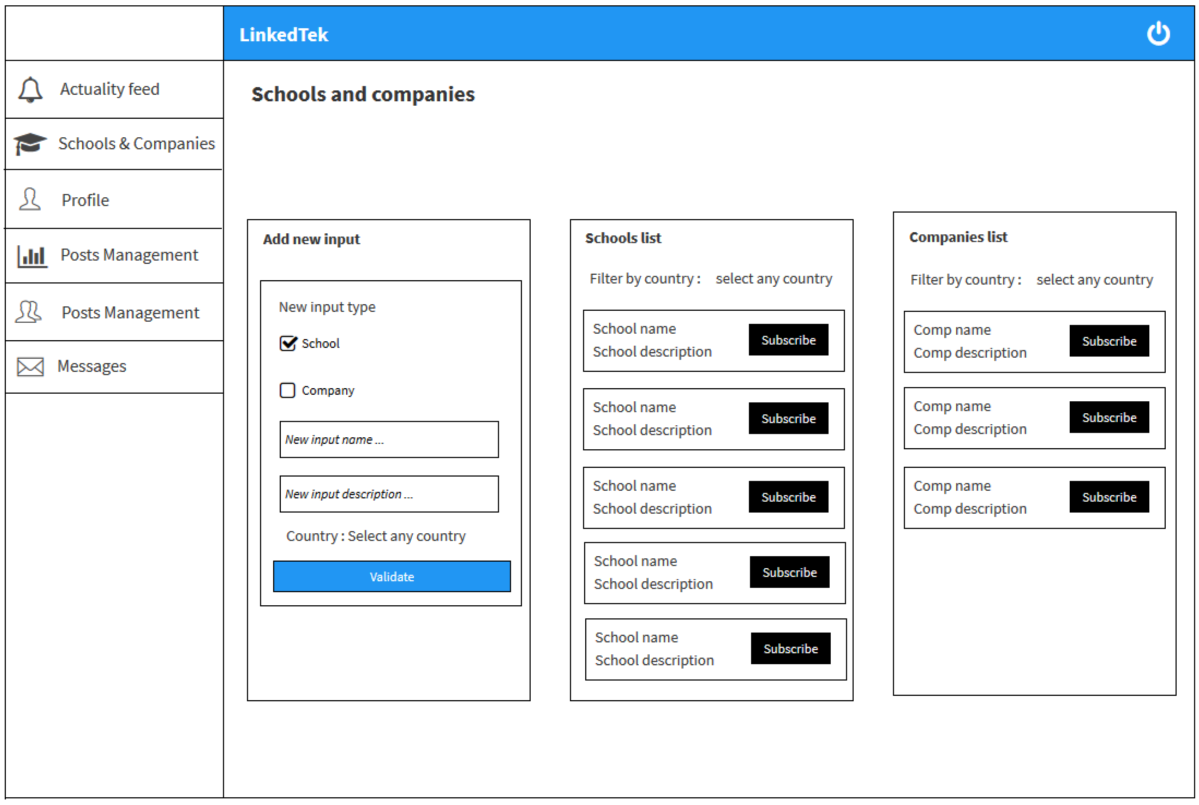


Figure 3-17 : Frontend schools and companies mockup

4. Backend architecture

A. General overview

The LinkedTek backend architecture is build thanks to the following micro-services:

- The API-Gateway
- The authentication micro-service
- The LinkedTek micro-service
- A mongo database
- A neo4j database.

Each of these micro-services are docker containers, which are orchestrated thanks to docker-compose, and the micro-service's transactions are done over the HTTP.

The API-Gateway, the authentication and the LinkedTek micro-services has been developed in javascript and are executed thanks to nodejs.

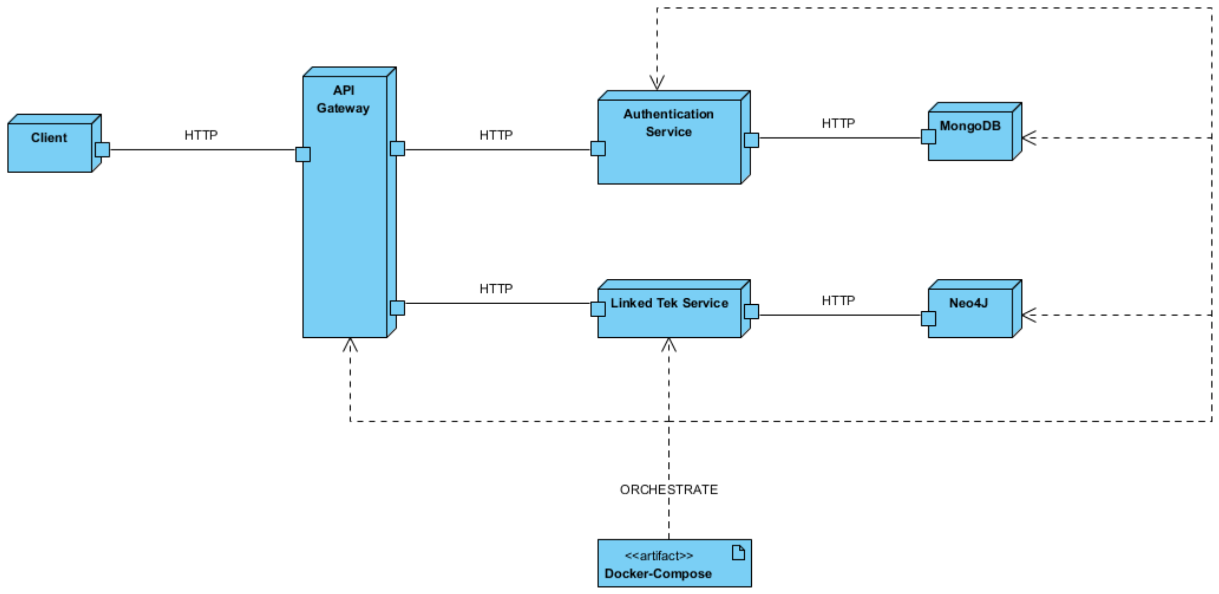


Figure 4-1 Back-end deployment diagram

B. The API-Gateway micro-service

The API-Gateway purpose is to ease the client queries by exposing a unique entry-point to the LinkedTek backend. The API Gateway then dispatch the client's queries to the wanted services.

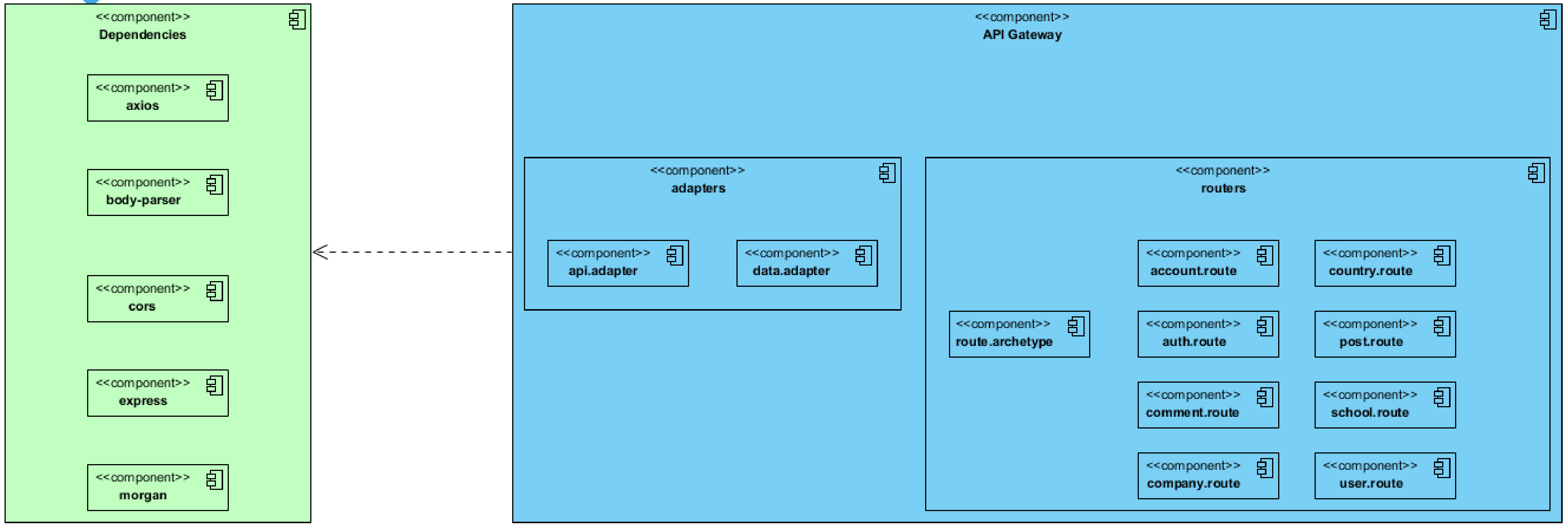


Figure 4-2API Gateway component diagrams

As shown on the Figure 4-2 the API-Gateway depends on:

- The axios library to send HTTP requests
- The express, cors and body-parser libraries to handle HTTP requests (reception and response)
- The morgan library to log any HTTP interaction

The API-Gateway is composed by two modules, the adapters modules which group API and data adapters, and the routers module which handles the routing of the various HTTP requests.

a. API adapter

The API adapter has two purposes. One is to transform the error returned by the axios to a simpler error object. The second one is two create a query handler to the authentication service and the linkedtek-service.

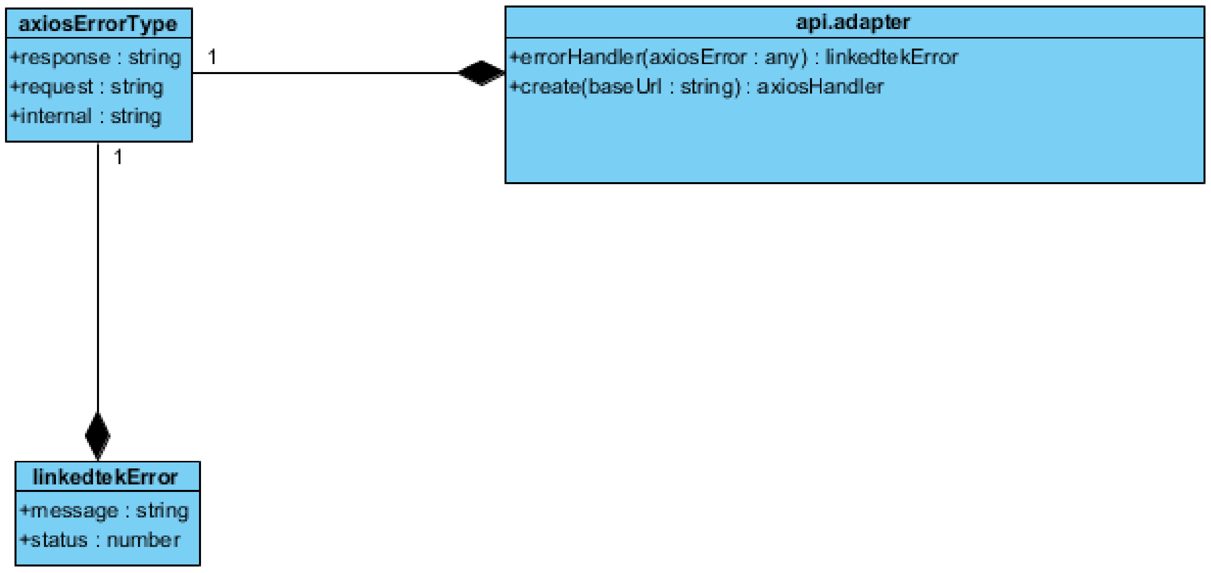


Figure 4-3API Adapter class diagram

b. Data adapter

The data-adapter purpose is to transform the response returned by both the authentication service and the linked-tek service to json data, that are then send to the client.

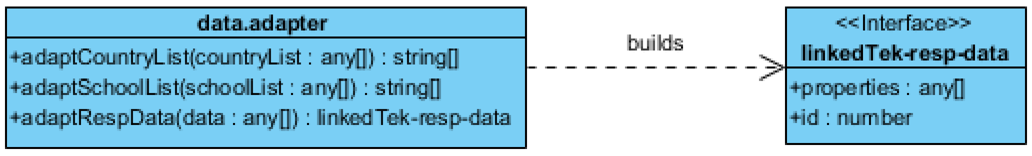

data.adapter
+adaptCountryList(countryList : any[]) : string[]
+adaptSchoolList(schoolList : any[]) : string[]
+adaptRespData(data : any[]) : linkedTek-resp-data
builds
<<Interface>>
linkedTek-resp-data
+properties : any[]
+id : number

Figure 4-4Data Adapter class diagram

c. Router activity[7]

All the API-Gateway routers have similar purposes, namely they dispatch the incoming request from the client to the adequate services, and vice-versa.

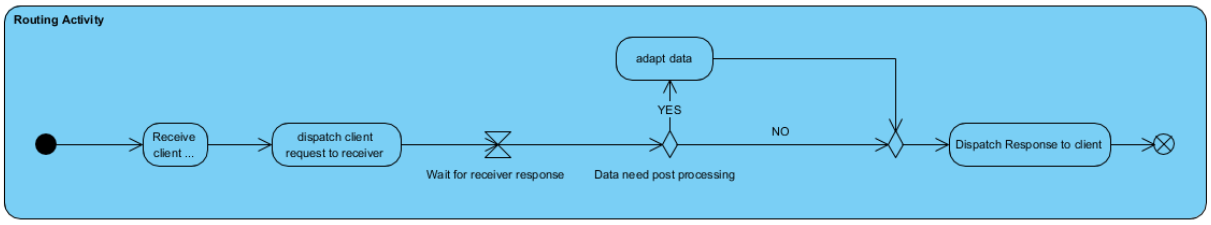

Routing Activity
Receive client ...
dispatch client request to receiver
Wait for receiver response
adapt data
YES
Data need post processing
NO
Dispatch Response to client

Figure 4-5route archetype activity

[7] The route api is described in the howto.md that can be found in the back-end repository.

C. The authentication micro-service

The authentication micro-services ensure the user authentication, it uses a mongo container to store the user credentials data. Obviously, the user’s password is never stored as is in the database, but an hash algorithm is used to authenticate an user which provides the correct password.

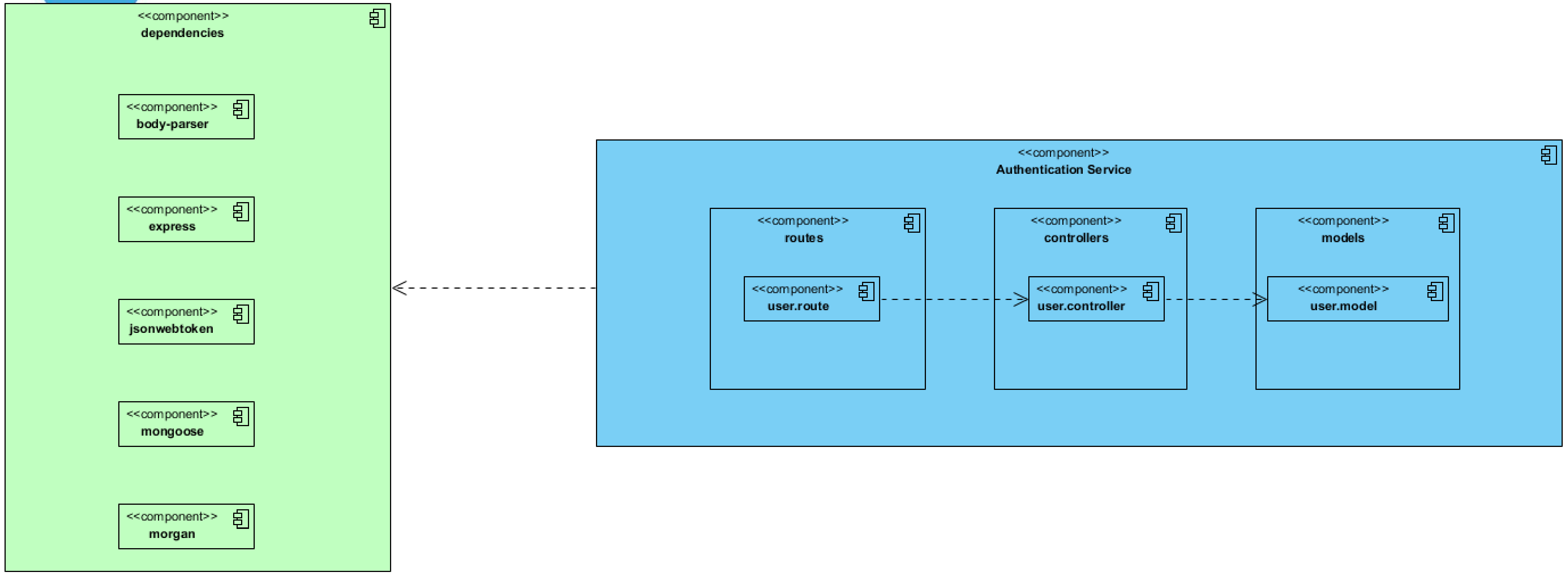


Figure 4-6Authentication service component diagrams

As shown of Figure 4-6 the authentication service depends on:

- The body-parser and express library to handle HTTP requests and responses.
- The mongoose library which is Object Document Mapper for the mongo database.
- The jsonwebtoken library to generates user's web token.
- The morgan library to log the HTTP transactions.

The authentication micro-service is composed by 3 components, the routers which handles the HTTP requests and responses, the controllers which manage any action on the database and the models which represents the data found in the mongo database.

a. User model

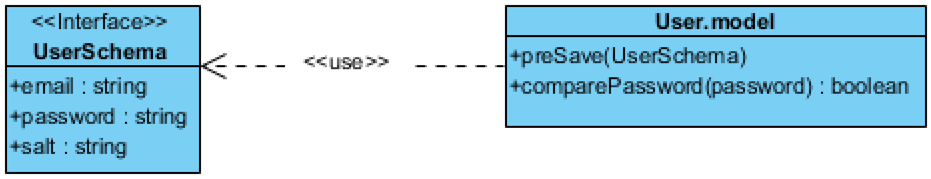
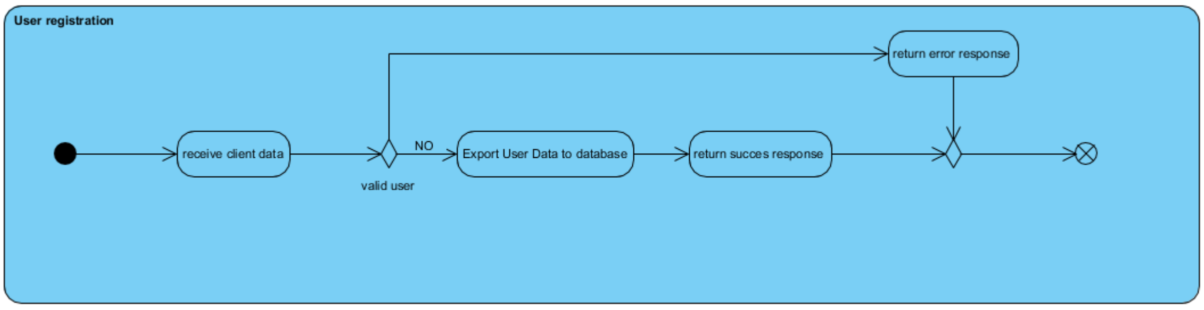
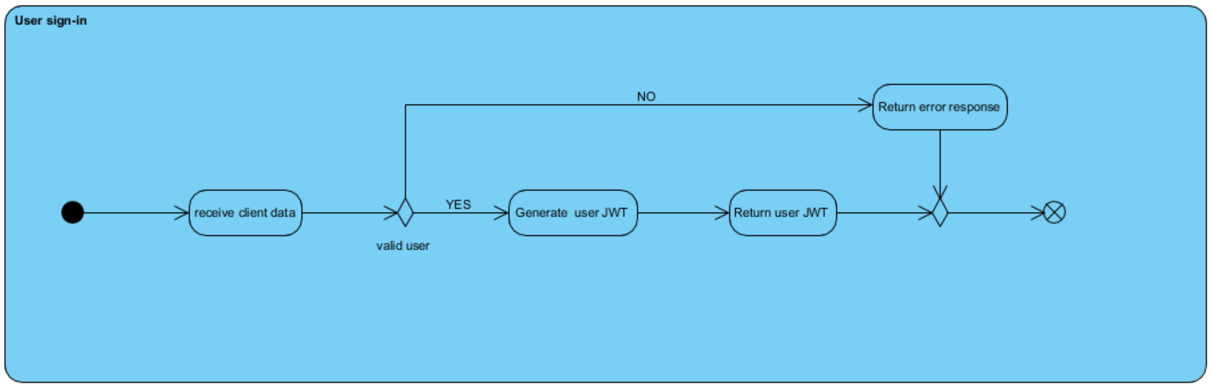


Figure 4-7UserModel class diagram

b. User authentication activities





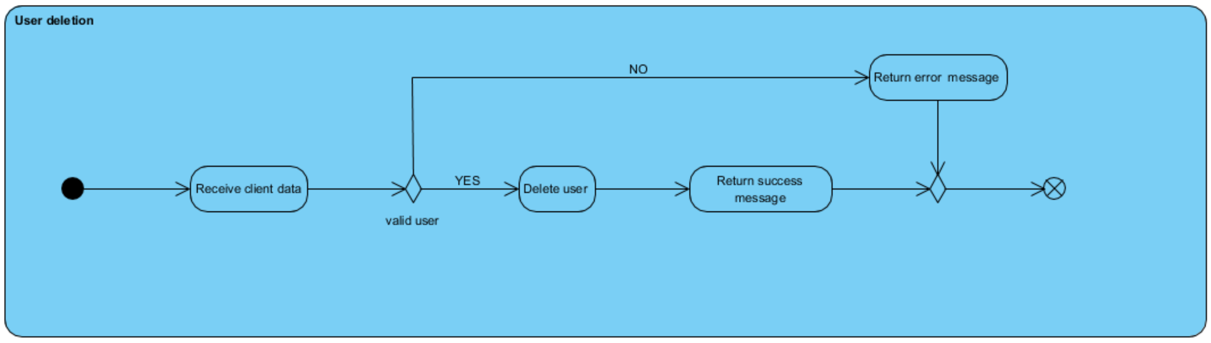


Figure 4-8Authentication service activity diagrams

As shown on Figure 4-8 a user can:

- Register itself
- Sign in and receive a jwt
- Delete its own account

D. The Linked-Tek micro-service

The Linked-Tek micro-service ensure the core functionalities of the Linked-Tek project. It is this service that is responsible of the creation, recuperation, update and deletion of the relational data (user, company, schools, etc…).

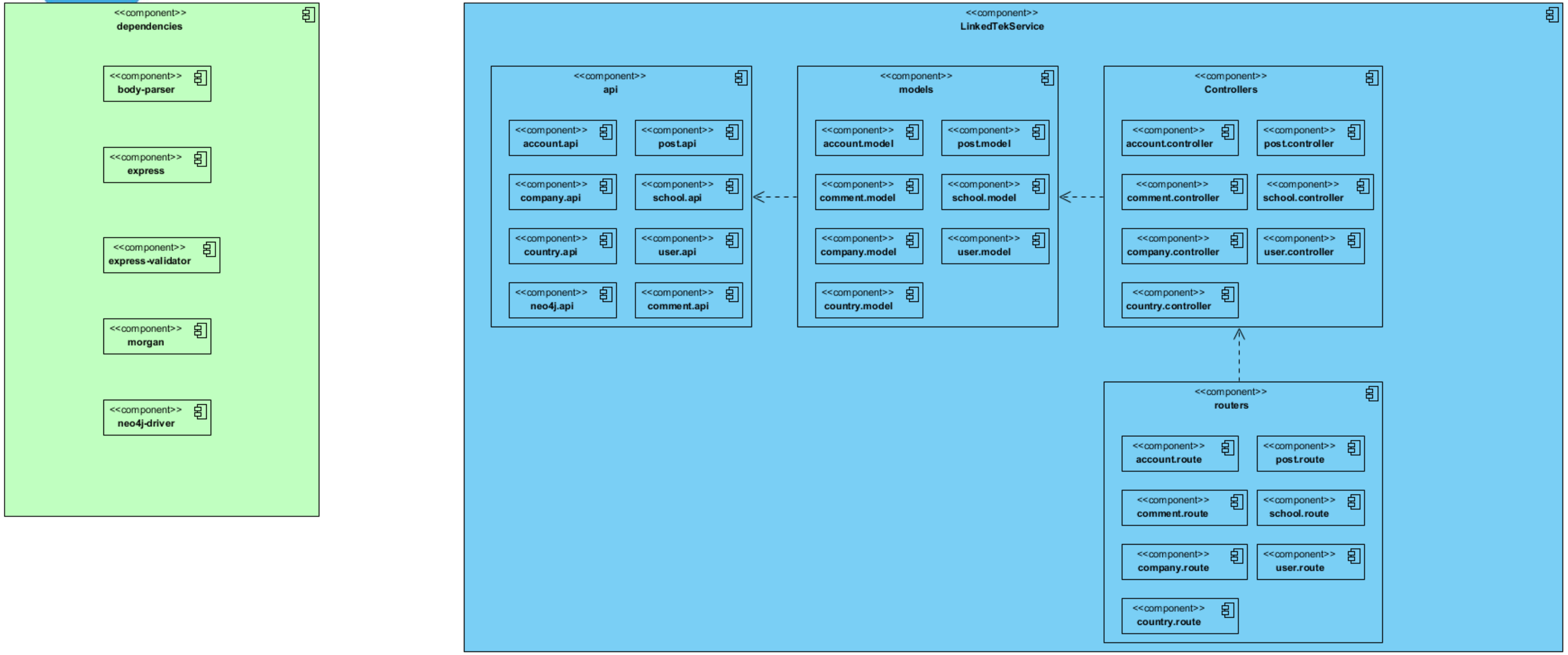


Figure 4-9Linked-Tek service component diagram

As shown on Figure 4-9 the linked-tek micro-services depends on:

- The body-parser, express, express-validator libraries to deal with the HTTP requests and responses.
- The neo4j-driver library to manage the neo4j database.
- The morgan library to log any http requests.

The linked-tek micro-services are build on 4 type of component, the routers which dispatch the HTTP requests to the appropriate controller, the controllers which check the data query data validity and execute the database operations thanks to the model components. The api components are façades between the model and the database operations.

a. LinkedTek Service activity overview

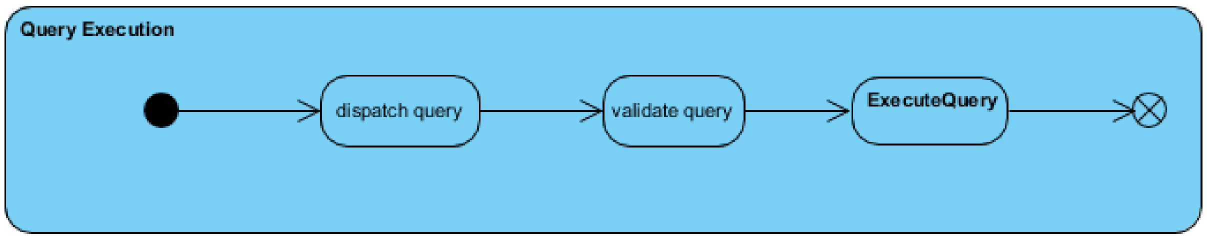


Figure 4-10LinkedTek Activity

The LinkedTek service activity is resumed by the Figure 4-10:

- The dispatch query activity is done by the router components.
- The validate and execute query are done by the controllers.

The following sections represents the core functionality of the LinkedTek service.

b. Account API

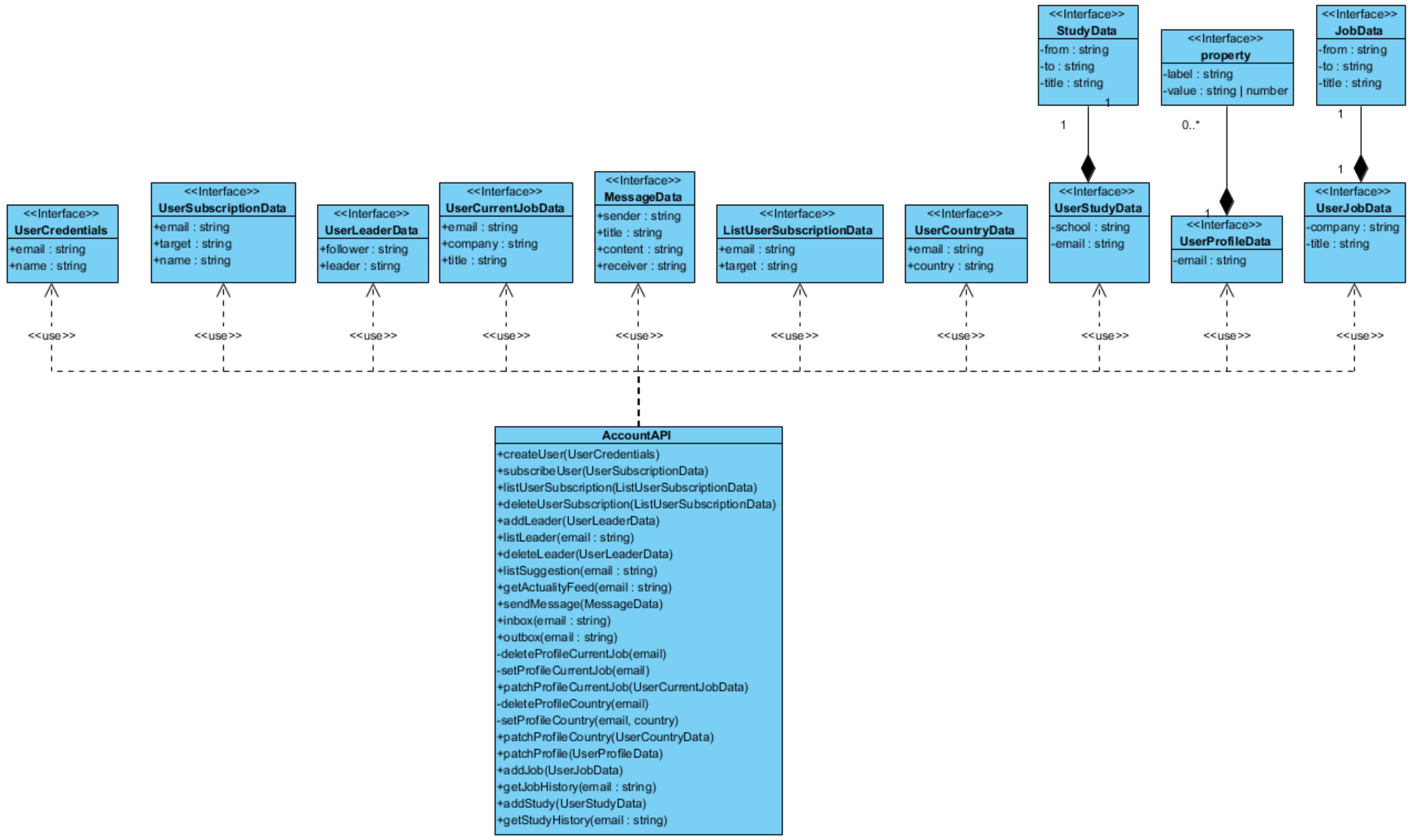


Figure 4-11Account API class diagram

Method	Purpose
createUser	Create a user in the database
subscribeUser	Subscribe a user to a company or a school
listUserSubscription	Get the user subscription list
deleteUserSubscription	Delete a user subscription
addLeader	Add a leader to the user
deleteLeader	Delete a user's leader
listSuggestion	List the leader, school and company suggestion
getActualityFeed	List the actuality of a user (posts and comments)
sendMessage	Send a message to another user
inbox	List the received message of a user
outbox	List the sent message of a user
patchProfileCurrentJob	Update the current job of a user
patchProfileCountry	Update the current country of a user
patchProfile	Update the profile data of a user (name, age)
addJob	Add a job to the job history of a user
getJobHistory	List the job history of a user
addStudy	Add a study to the user profile
getStudyHistory	List the user studies

The account manages the user account relations, to:

- another user leader/follower relationship
- school and company subscription
- message to/from another users
- current job and country
- job and study history

c. Post API

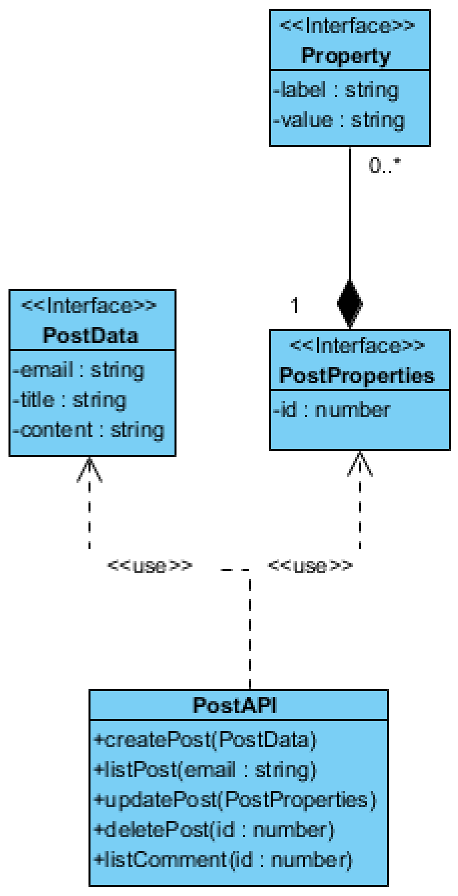


Figure 4-12Post API class diagram

The post API is used by the user who wants to publish a post.

d. Comment API

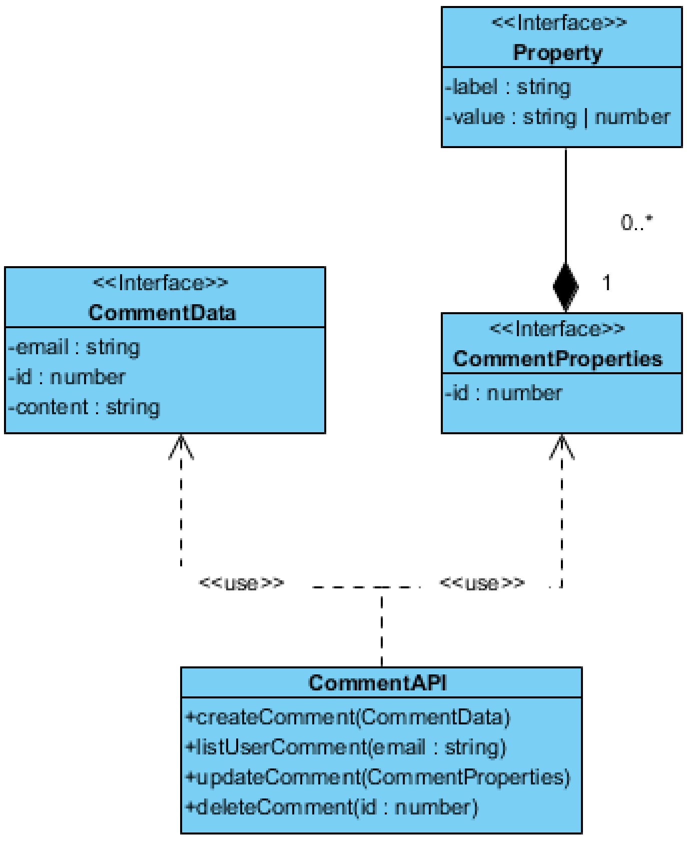


Figure 4-13Comment API class diagram

The comment API is used by the user who wants to comment a post.

e. Company API

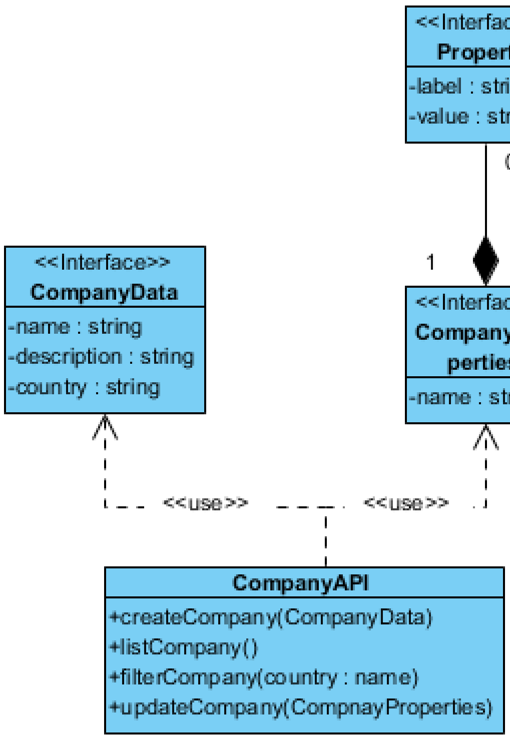


Figure 4-14Company API

The company API is used by the users who wants to create or update a company. But also, to list the existing company and to list the existing company by country.

f. School API

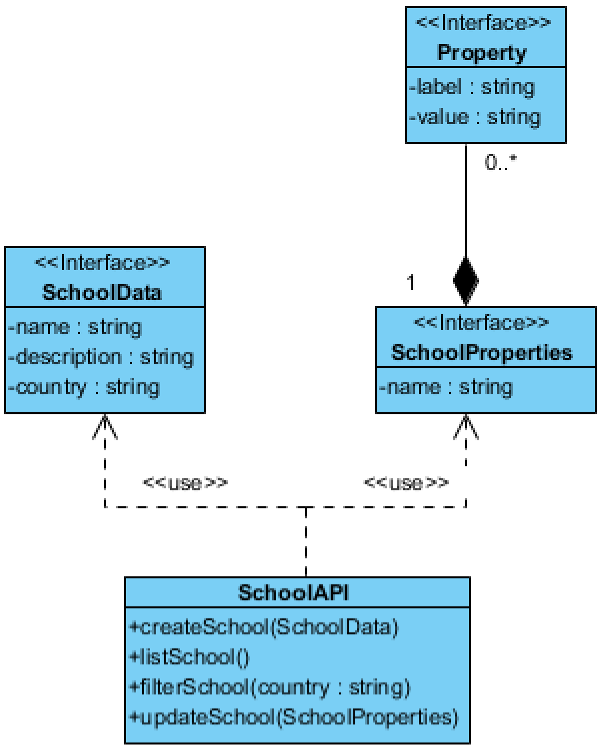


Figure 4-15School API

The school API is used by the users who wants to create or update a school. But also, to list the existing school and to list the existing school by country.

g. User API

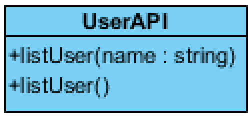


Figure 4-16User API class diagram

The user API is used to list the user by name, or all the existing user.

h. Country API

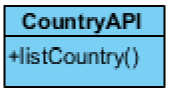


Figure 4-17Country API class diagrams

The country API is used to list the country stored in the database.

i. Neo4j API

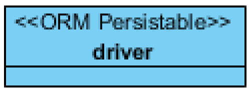


Figure 4-18Neo4j api

The neo4j API exposes the neo4J driver function, it is used by the other API components.

j. Controllers activity

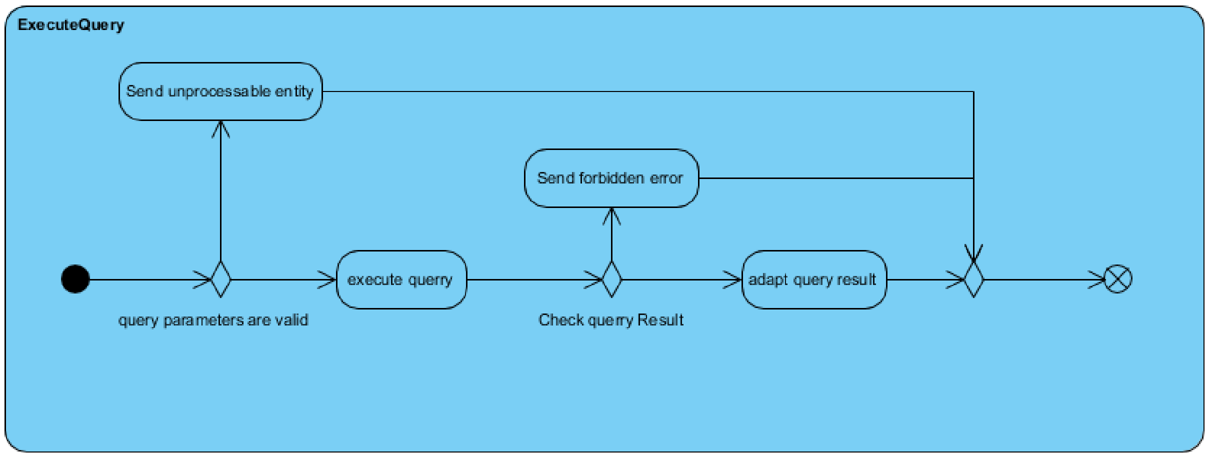


Figure 4-19Controller activity diagrams

Each API is used by the controllers via the models components, following the Figure 4-19.

E. Neo4j micro-service

Neo4j is a graph database management system developed by Neo4j.inc. It is an ACID-compliant transactional database with native graph storage and processing.

a. Database architecture

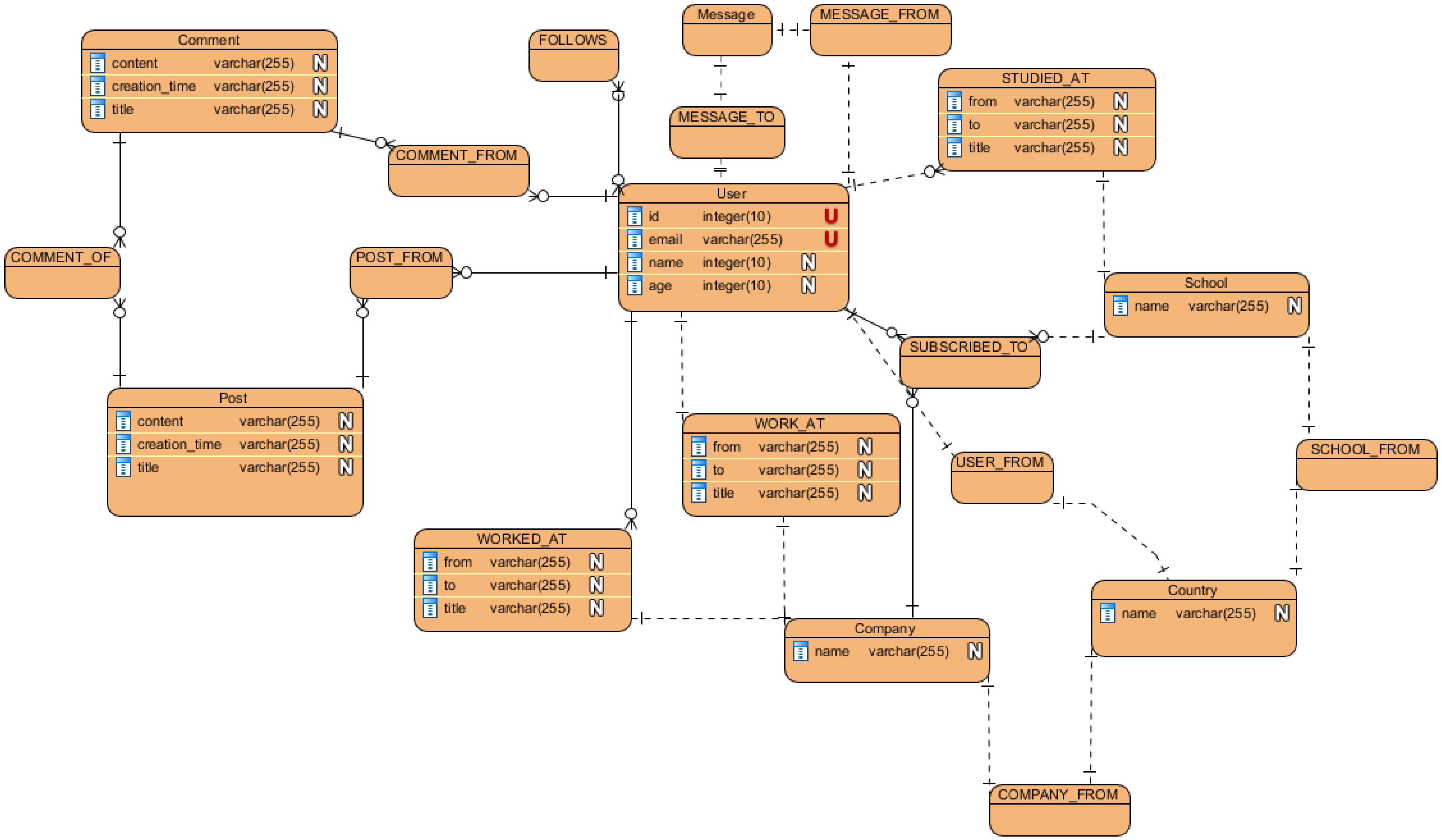


Figure 4-20Neo4j ORM graph

b. Administration

The administration of the database is done thanks to the neo4J administration page, where an authenticated administrator has the full power on the database. You can find information on the neo4j database administration on the neo4j database site.

5. Traceability matrix

This matrix makes the correspondence between components, classes, functions and requirements developed in the request for proposal.

Id requirement	Requirement description	Component	Function / action
REQ_DESIGN_010	You need to release the whole platform (databases included) as a docker	Backend: ALL	
REQ_DESIGN_020	You must expose a JSON Rest API on the back	Backend: - API Gateway - Authentication micro-service - LinkedTek micro-service	
REQ_DESIGN_030	You must, at least, respect the Level 2 of the Richardson maturity model	Backend: - API Gateway - Authentication micro-service - LinkedTek micro-service	
REQ_DESIGN_040	You must provide a functional and well-designed authentication system	Backend: - Authentication micro-service	
REQ_DESIGN_050	You must choose between NodeJS, python, Golang, PHP, java for the backend part	Backend: ALL	
REQ_DESIGN_060	You must provide a JavaScript based frontend (you are free on the JavaScript framework choice)	Frontend: ALL	
REQ_DESIGN_070	You must have a persistent storage	Backend: - Mongo-db	

	for all the data (the data needs to stay stored, even after a relaunch of the containers)	- Neo4j	
REQ_DESIGN_080	You must provide a docker-compose file (docker-compose.yml) with a fully working "docker-compose up" command.	Backend: Docker-compose.yml	
REQ_DESIGN_090	You must be able to manage two kinds of user: administrator and classic user	Backend: - Neo4j	DE FACTO: User use linked tek client and administrators use neo4j administration console.
REQ_DESIGN_100	You must provide a fully tested project, on both part: back and front	ALL	
REQ_DESIGN_110	You must provide several micro-services; the project needs to be cut under several different micro-services (for scalability and resilience purposes)	Backend: ALL	
REQ_DESIGN_120	Each micro-service needs to be independently scalable of the others (we need to be able to run 5 instances of the user micro-service if we want but only 1 school micro-service for example)	Backend: ALL	
REQ_DESIGN_130	The micro-services communication needs to be language agnostic (JSON / GRPC / …)	Backend: ALL	

REQ_FUNC_010	You must be able to register / log-in on the software	Frontend:
		Login.js
		Register.js
		
		Backend:
			Frontend:
			clickOnRegisterButton()
			clickOnSubmitButton()
			
			Backend:

		Authentication micro-service	Gateway.auth.api

REQ_FUNC_020	You must be able to add / edit / list the schools	Frontend:
		Schools and companies.js
		
		Backend:
		LinkedTek micro-service
			Frontend:
			componentWillMount()
			addNewSchoolOrCompany()
			handleSchoolOrCompanySubscription()
			handleEditCompanyModalCloseValidated()
			handleEditSchoolModalCloseValidated ()
			
			Backend:
			School.api

REQ_FUNC_030	You must be able to add / edit / list the companies	Frontend:
		Schools and companies.js
		
		Backend:
		LinkedTek micro-service
			Frontend:
			componentWillMount()
			addNewSchoolOrCompany()
			handleSchoolOrCompanySubscription()
			handleEditCompanyModalCloseValidated()
			handleEditSchoolModalCloseValidated ()
			
			Backend:
			Company.api

REQ_FUNC_040	As administrator, you must be able to list all the users	Frontend: Neo4j Backend: Neo4j	Frontend: Backend:
REQ_FUNC_050	As administrator, you must be able to edit / remove /add/ ban the users	Frontend: Neo4j Backend: Neo4j	Frontend: Backend:
REQ_FUNC_060	As administrator, you must be able to delete schools or companies	Frontend: Neo4j Backend: Neo4j	Frontend: Backend:
REQ_FUNC_070	As administrator, you must be able to edit / remove / add all the posts and user's comments.	Frontend: Neo4j Backend: Neo4j	Frontend: Backend:
REQ_FUNC_080	A user (non-administrator) is not able to remove school or companies	Frontend: *Schools and companies.js* Backend:	Frontend: *Not applicable* Backend: *Not applicable*

		Linked-tek micro-service	
REQ_FUNC_090	A user can subscribe or unsubscribe in several companies or in several schools	Frontend: *Schools and companies.js* Backend: Linked-tek micro-service	Frontend: *handleSchoolOrCompanySubscription()* Backend: Account.subscription.api
REQ_FUNC_100	A user can see other users' profiles (schools, companies, ...)	Frontend: *Relations.js* Backend: Linked-tek micro-service	Frontend: *handleRelationModalCloseValidated()* Backend: School.api
REQ_FUNC_110	A user can add or remove another user to is relations	Frontend: *Relations.js* Backend: Linked-tek micro-service	Frontend: *handleRemoveRelation()* Backend: Account.leader api
REQ_FUNC_120	A user has access to a feed with all is relations' activities (new post, comment, ...)	Frontend: *Actualities.js* Backend: Linked-tek micro-service	Frontend: *componentWillMount()* Backend: Account.feed.api

REQ_FUNC_130	A user can add a new post, edit or remove it	Frontend: *Posts.js*
		
		Backend:
			Frontend: *componentWillMount()*
			handleRemovePost()
			handleEditPost()
			handleNewPost()
			
			Backend: Post.api

REQ_FUNC_140	A user can comment a post (and edit / remove the comment as well)	Frontend: *Posts.js*
		
		Backend: Linked-tek micro-service
			Frontend: *componentWillMount()*
			handleEditComment()
			handleRemoveComment()
			addNewComment()
			
			Backend: Comment.api

REQ_FUNC_150	The platform needs to suggest to the user new relation (thanks to common markers like	Frontend: *Relations.js* Backend: Linked-tek micro-service	Frontend: *componentWillMount()* Backend: Account.suggestions.api

	schools, companies or relations)		

REQ_FUNC_160	A user can send, see and respond to messages from another user.	Frontend:
		Message.js
		
		Backend:
		Linked-tek micro-service
			Frontend:
			componentWillMount()
			sendNewMessage()
			handleSendReplyMessage()
			
			Backend:
			message.api