LinkedTek

T10 - SWA - Advanced Cloud

« Advanced Cloud »

Software Architecture Specifications

Table of contents

[Introduction 4](#_Toc10241430)

[1. Project context 4](#_Toc10241431)

[2. Global architecture 5](#_Toc10241432)

[3. Frontend architecture 5](#_Toc10241433)

[A. General overview 5](#_Toc10241434)

[B. Activities description 6](#_Toc10241435)

[a. Login 6](#_Toc10241436)

[b. Register 6](#_Toc10241437)

[c. Feed 6](#_Toc10241438)

[d. Actuality details 7](#_Toc10241439)

[e. Messages 8](#_Toc10241440)

[f. Posts management 8](#_Toc10241441)

[g. Profile 9](#_Toc10241442)

[h. Relations 10](#_Toc10241443)

[i. Schools and companies 11](#_Toc10241444)

[j. Left menu 12](#_Toc10241445)

[C. Frontend activity diagrams 13](#_Toc10241446)

[D. Frontend mock-ups design 21](#_Toc10241447)

[4. Backend architecture 25](#_Toc10241448)

[Component 1 25](#_Toc10241449)

[Component 2 25](#_Toc10241450)

[Component 3 25](#_Toc10241451)

[5. Traceability matrix 26](#_Toc10241452)

Figures List

[Figure 1 : Frontend login activity diagram 13](#_Toc10241413)

[Figure 2 : Frontend register activity diagram 13](#_Toc10241414)

[Figure 3 : Frontend actuality feed activity diagram 14](#_Toc10241415)

[Figure 4 : Frontend actuality details activity diagram 14](#_Toc10241416)

[Figure 5 : Frontend post management activity diagram 15](#_Toc10241417)

[Figure 6 : Frontend profile activity diagram 16](#_Toc10241418)

[Figure 7 : Frontend relations activity diagram 17](#_Toc10241419)

[Figure 8 : Frontend school and company activity diagram 18](#_Toc10241420)

[Figure 9 : Frontend messages activity diagram 19](#_Toc10241421)

[Figure 10 : Frontend menu activity diagram 20](#_Toc10241422)

[Figure 11 : Frontend actuality feed mockup 21](#_Toc10241423)

[Figure 12 : Frontend actuality details mockup 21](#_Toc10241424)

[Figure 13 : Frontend messages mockup 22](#_Toc10241425)

[Figure 14 : Frontend posts management mockup 22](#_Toc10241426)

[Figure 15 : Frontend user profile mockup 23](#_Toc10241427)

[Figure 16 : Frontend relations mockup 23](#_Toc10241428)

[Figure 17 : Frontend schools and companies mockup 24](#_Toc10241429)

# Introduction

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

# 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.

# 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 find at the 4 part of this document.

# Frontend architecture

For this project, the frontend part had been made using **JavaScript** language and more specifically, **Node.js[[1]](#footnote-1)** and the **React[[2]](#footnote-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 LinkTek project, the frontend uses as much as possible the **Material-UI[[3]](#footnote-3)** framework to design the components.

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

## 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]](#footnote-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]](#footnote-6)** commented.

## Activities description

### 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.

### 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.

### 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*

### 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*

### 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*

### 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*

### 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 :

### 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*

### 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*

### 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.

## Frontend activity diagrams

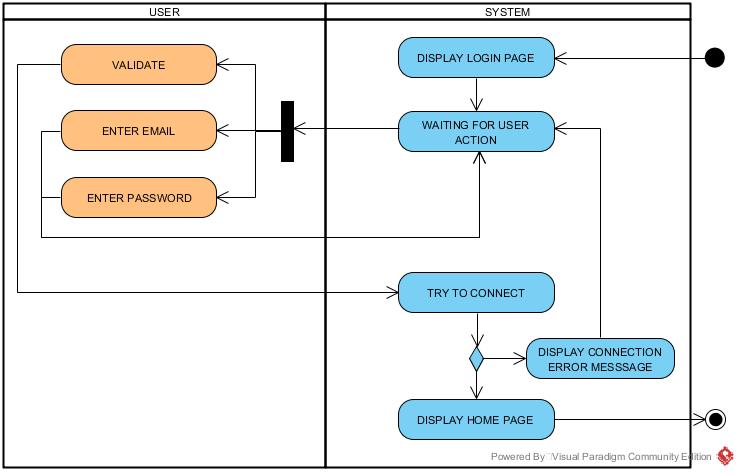


Figure 1 : Frontend login activity diagram

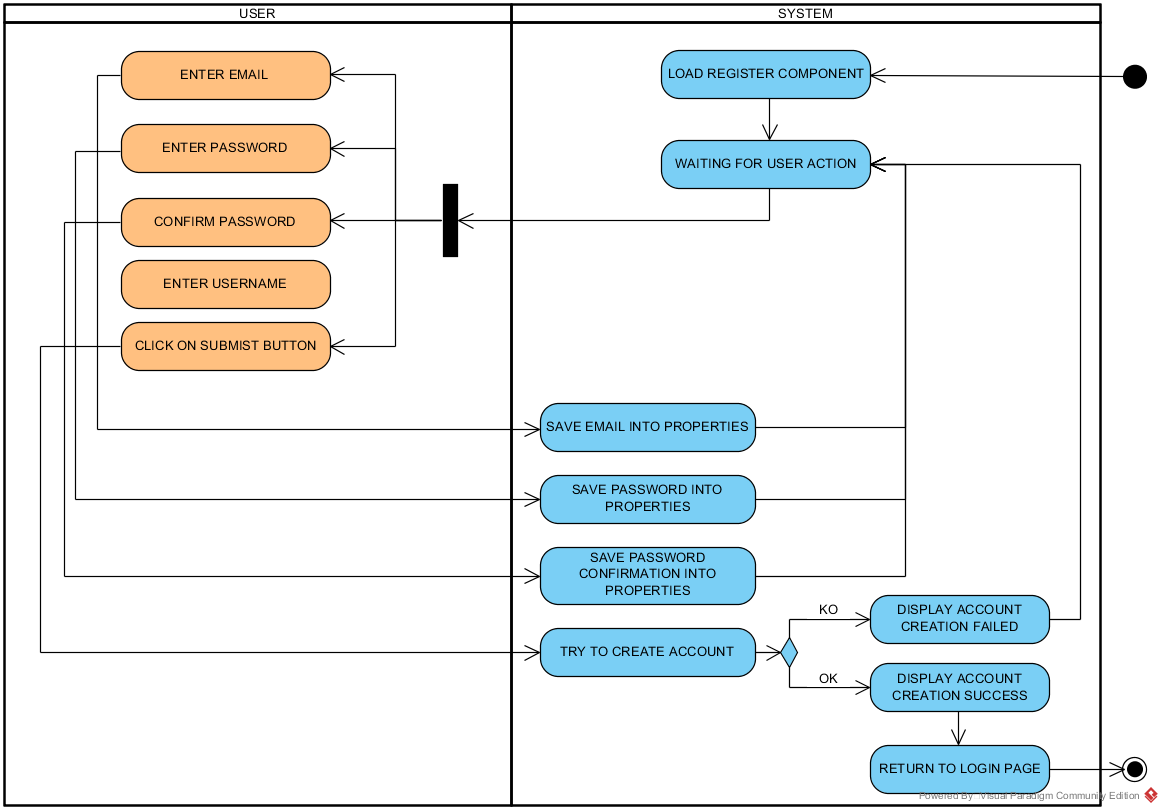


Figure 2 : Frontend register activity diagram

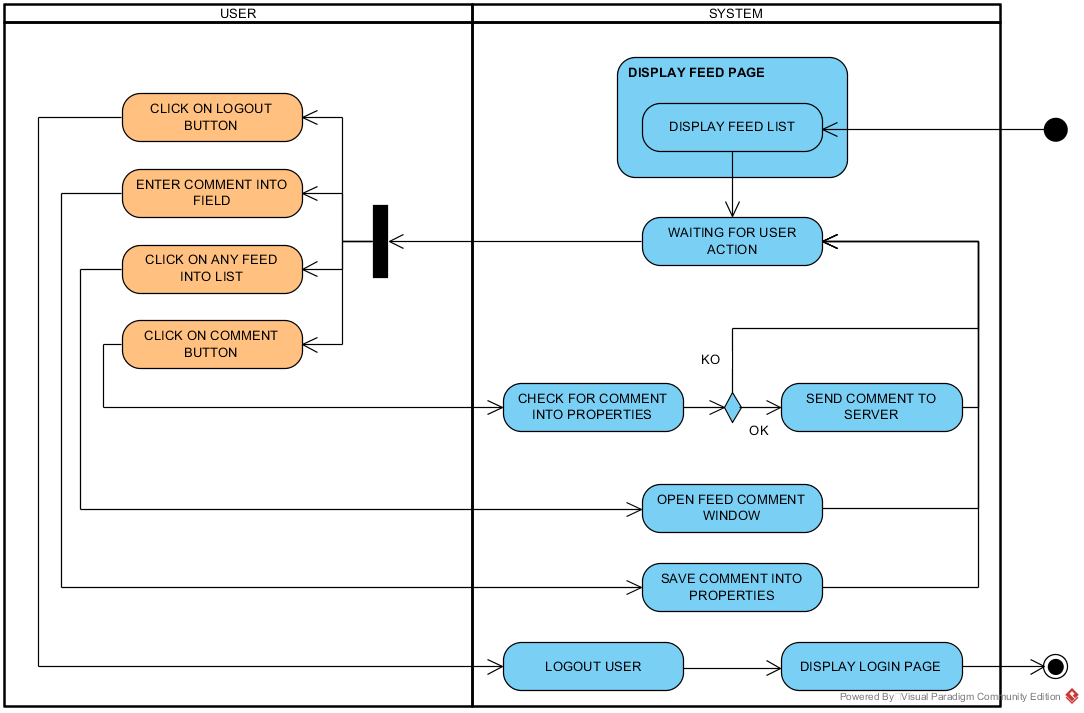


Figure 3 : Frontend actuality feed activity diagram

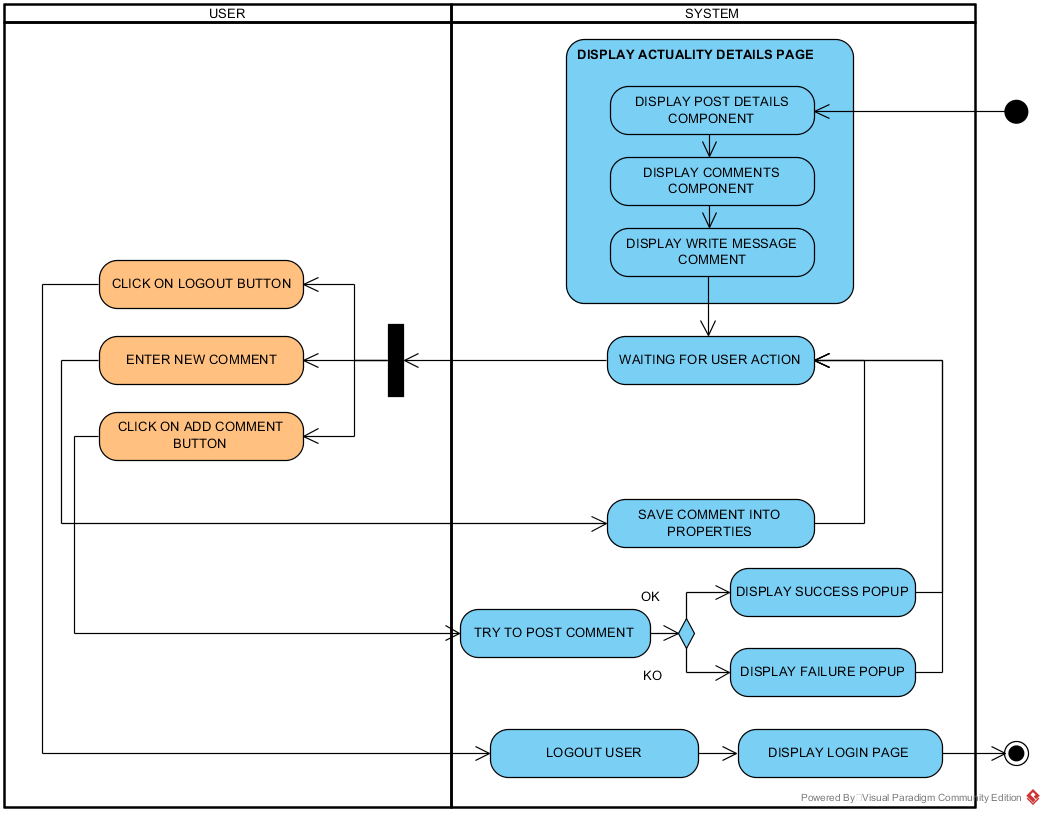


Figure 4 : Frontend actuality details activity diagram

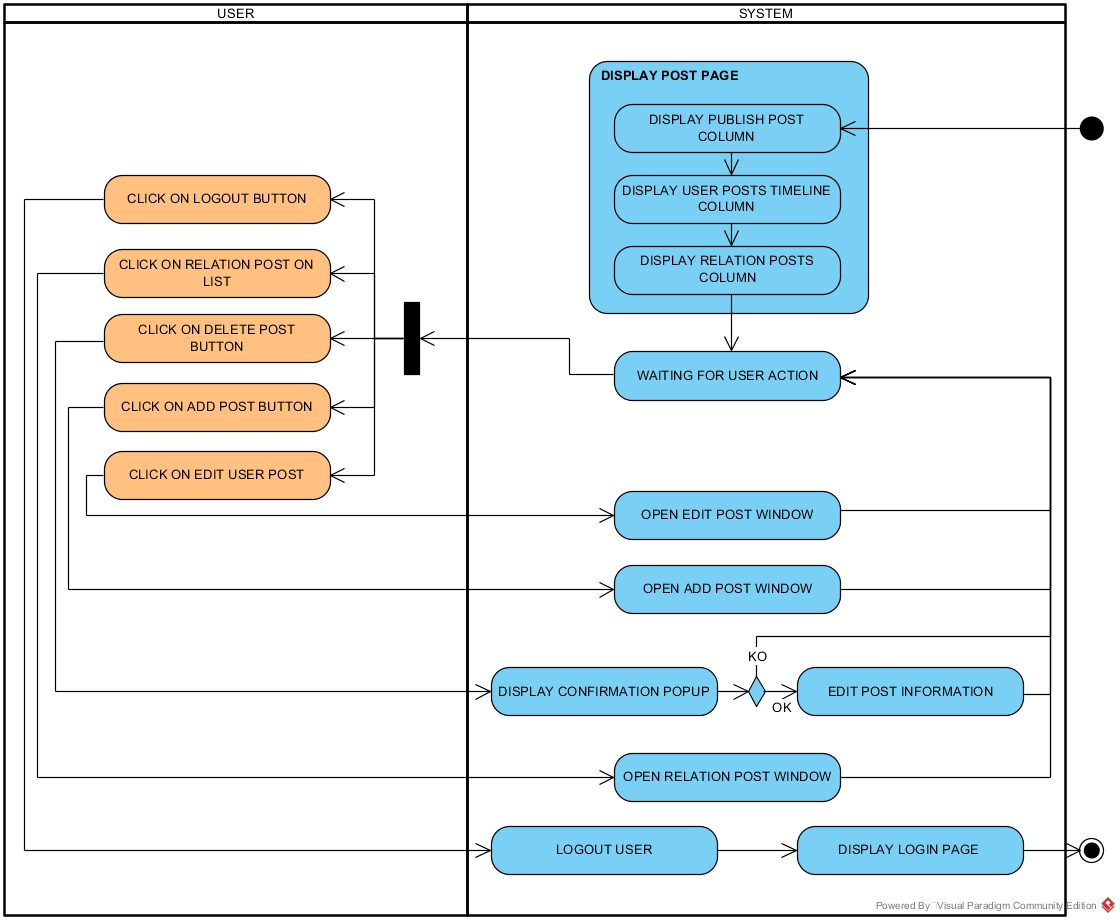


Figure 5 : Frontend post management activity diagram

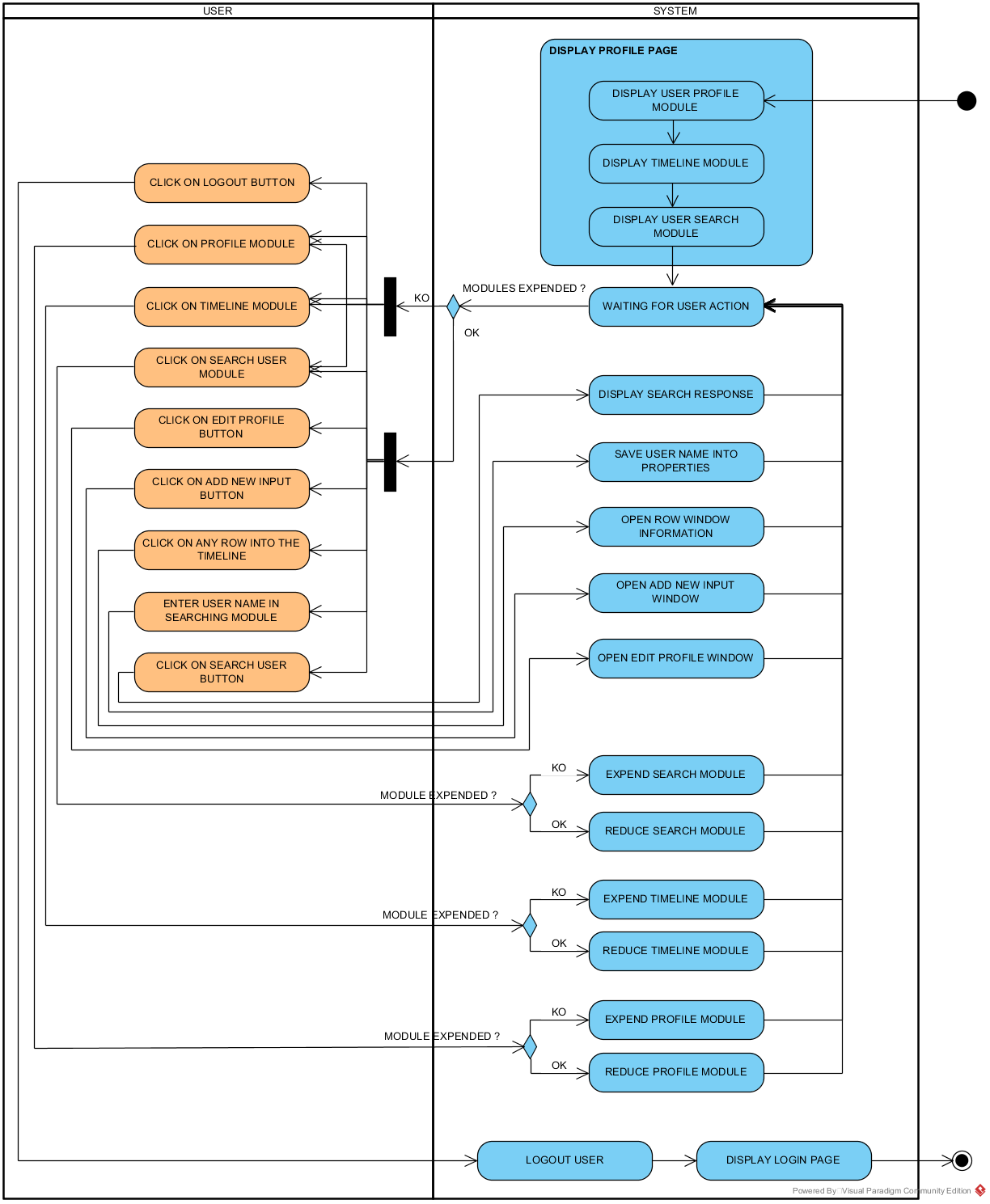


Figure 6 : Frontend profile activity diagram

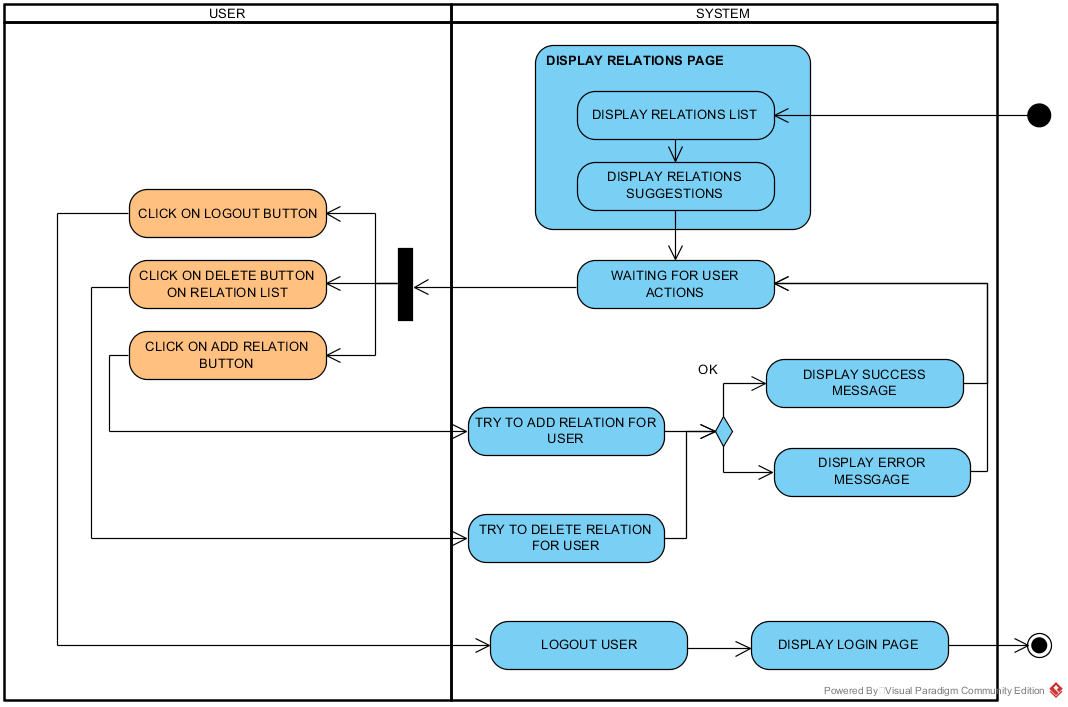


Figure 7 : Frontend relations activity diagram

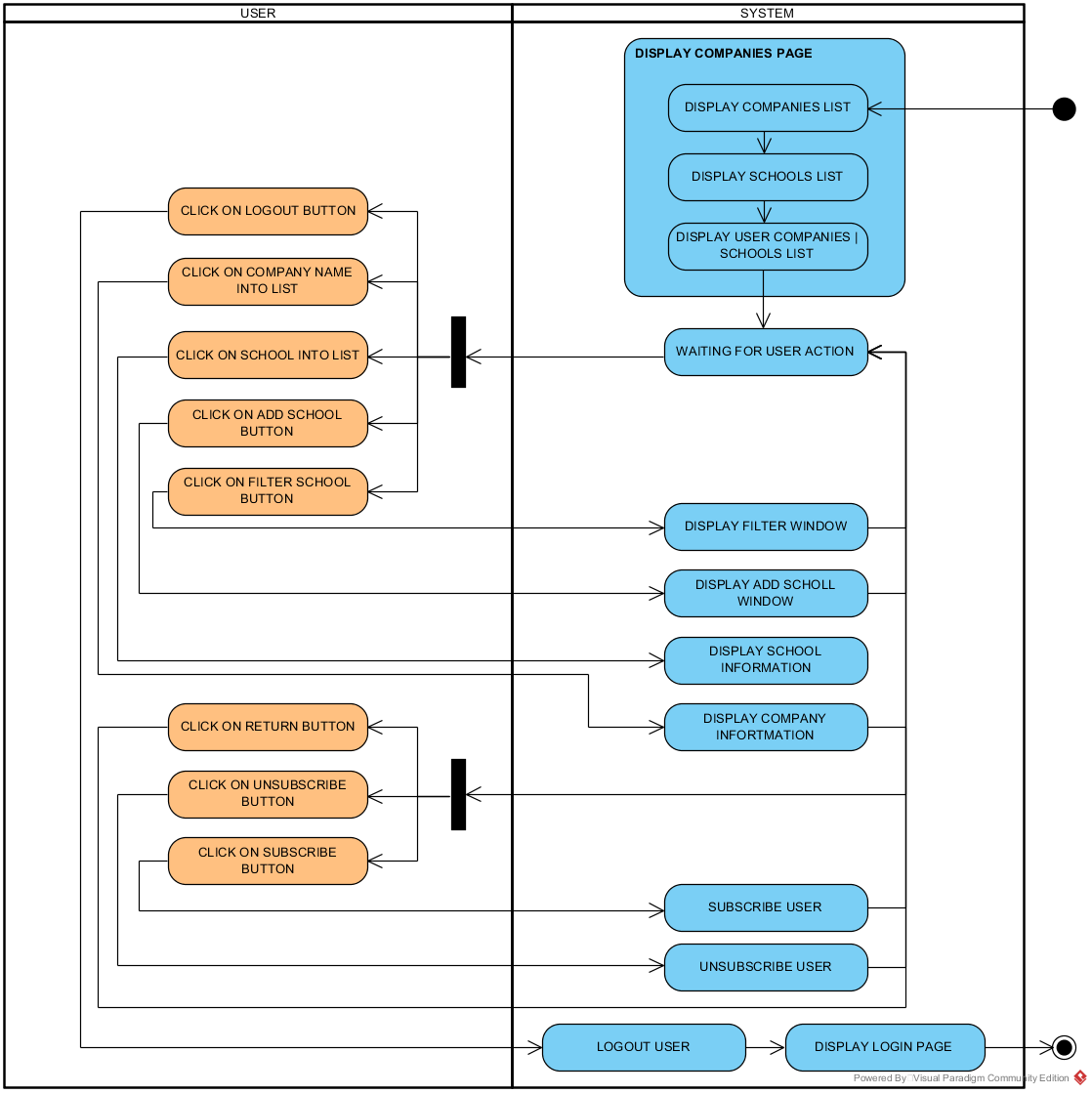


Figure 8 : Frontend school and company activity diagram

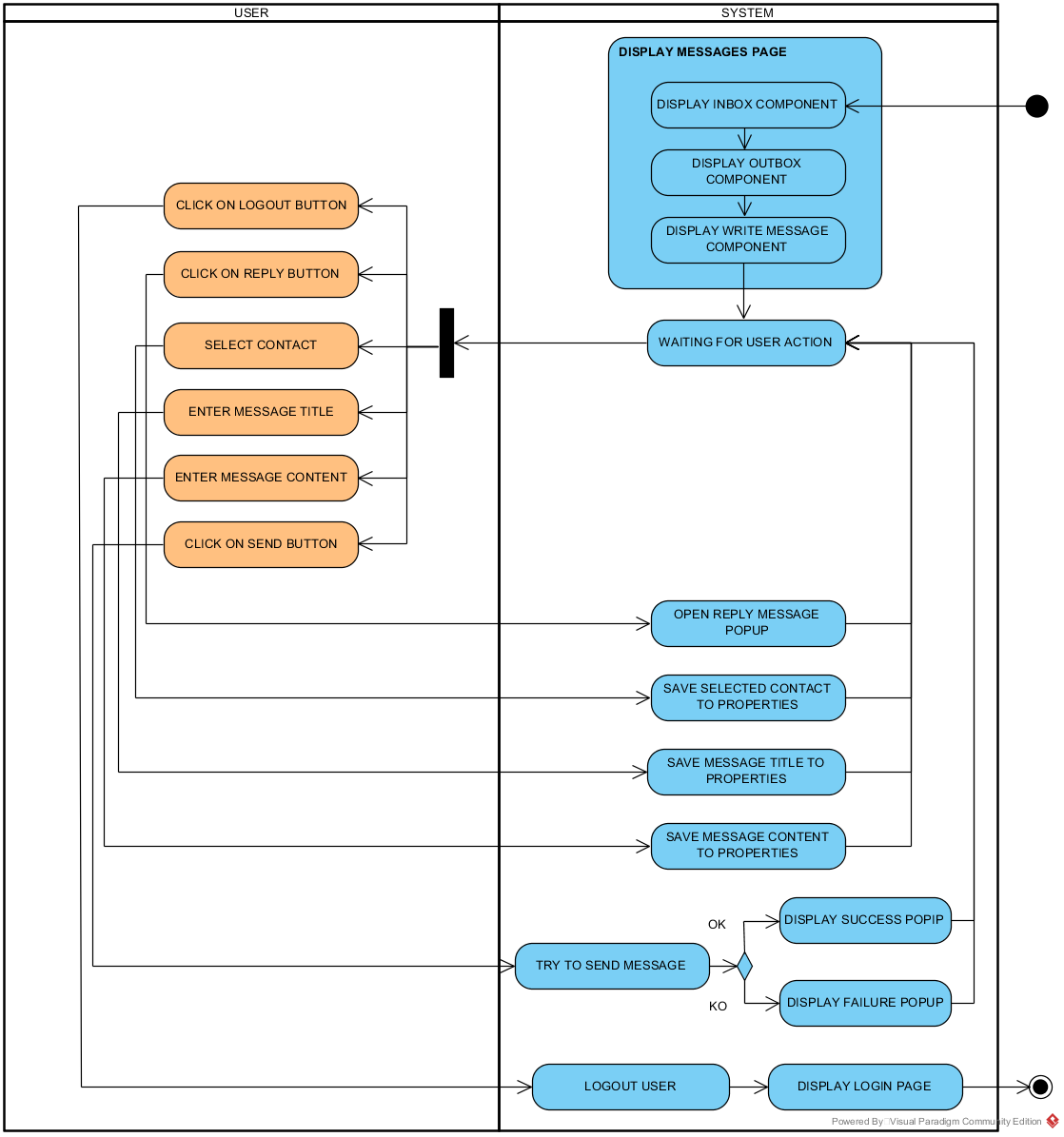


Figure 9 : Frontend messages activity diagram

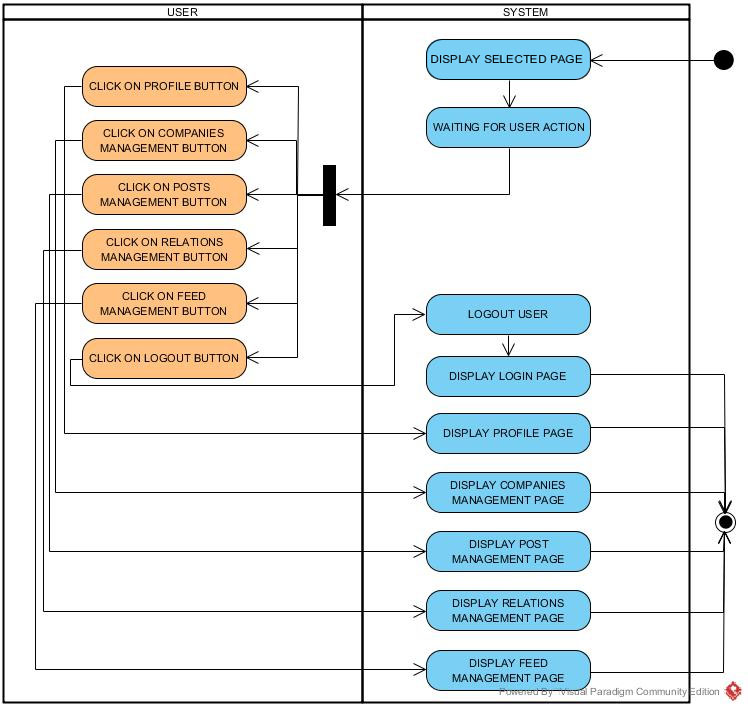


Figure 10 : Frontend menu activity diagram

## Frontend mock-ups design

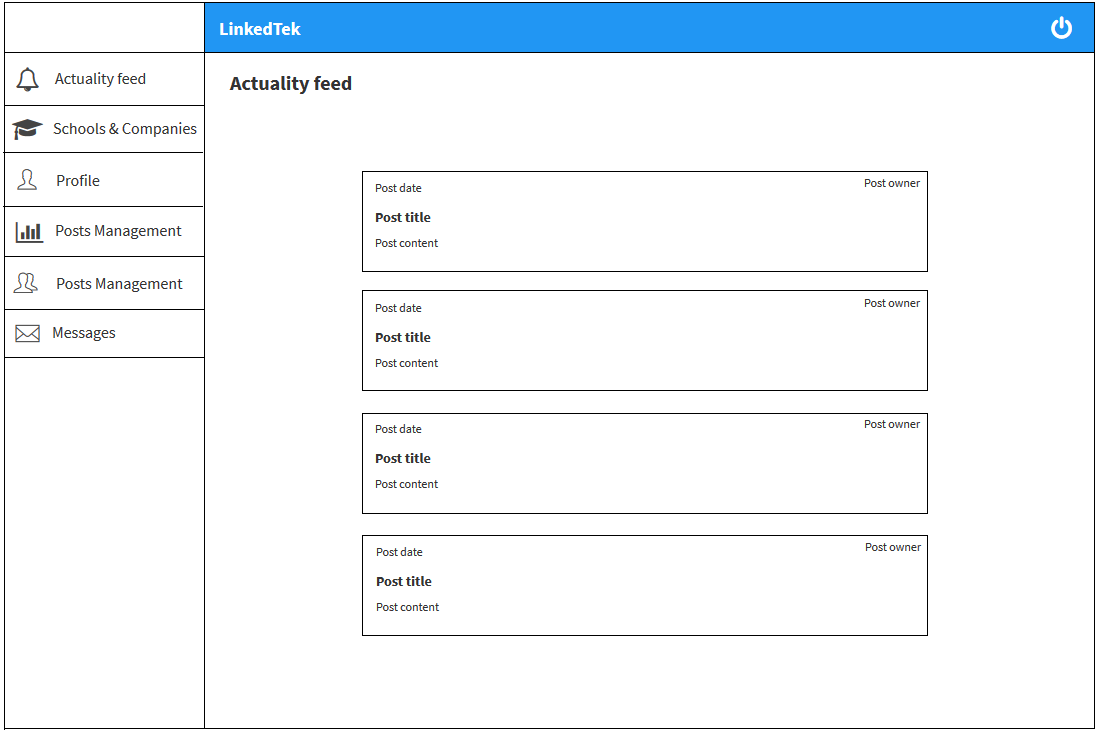


Figure 11 : Frontend actuality feed mockup

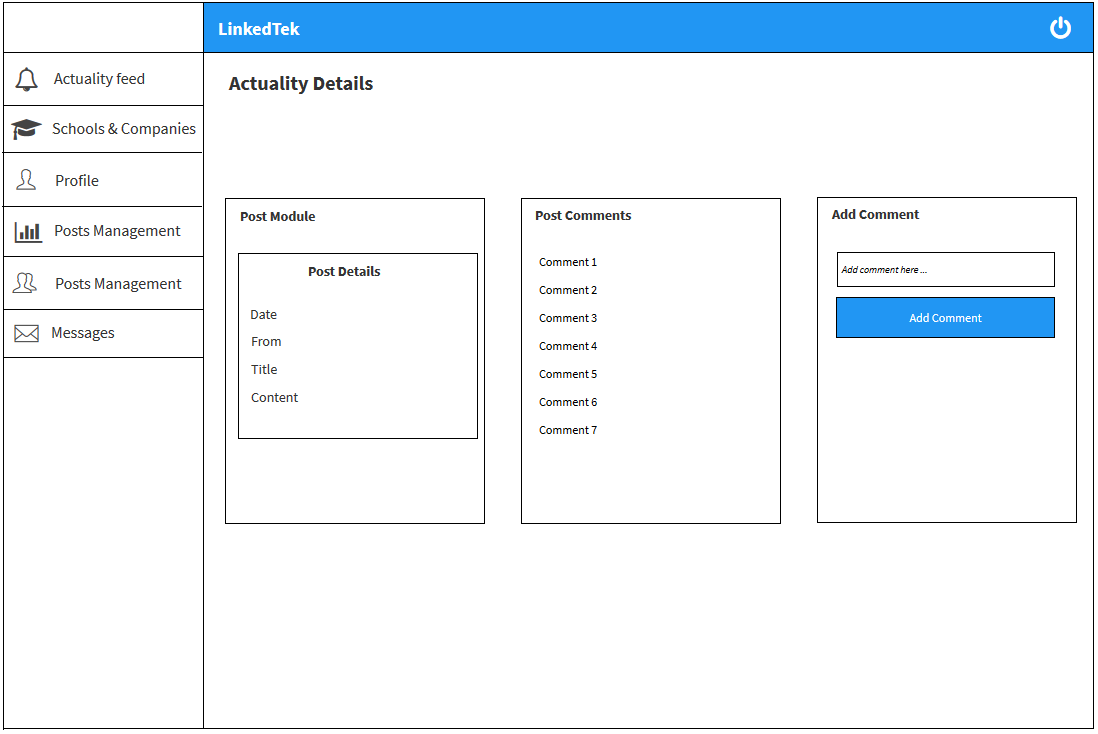


Figure 12 : Frontend actuality details mockup

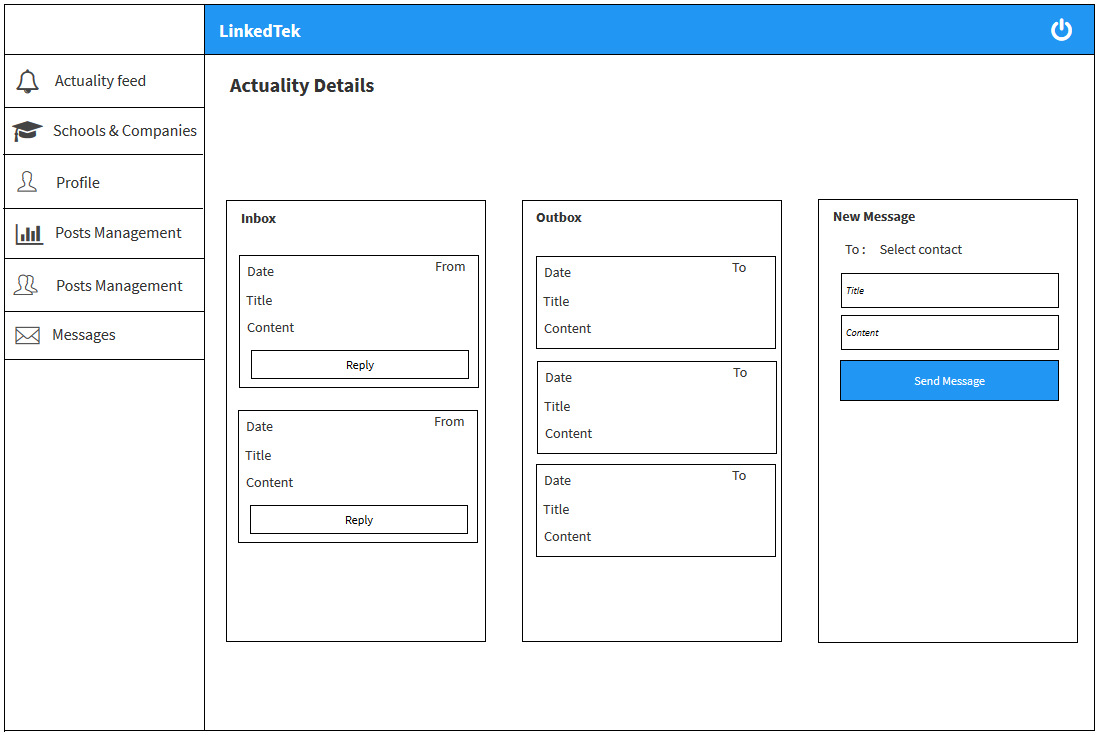


Figure 13 : Frontend messages mockup

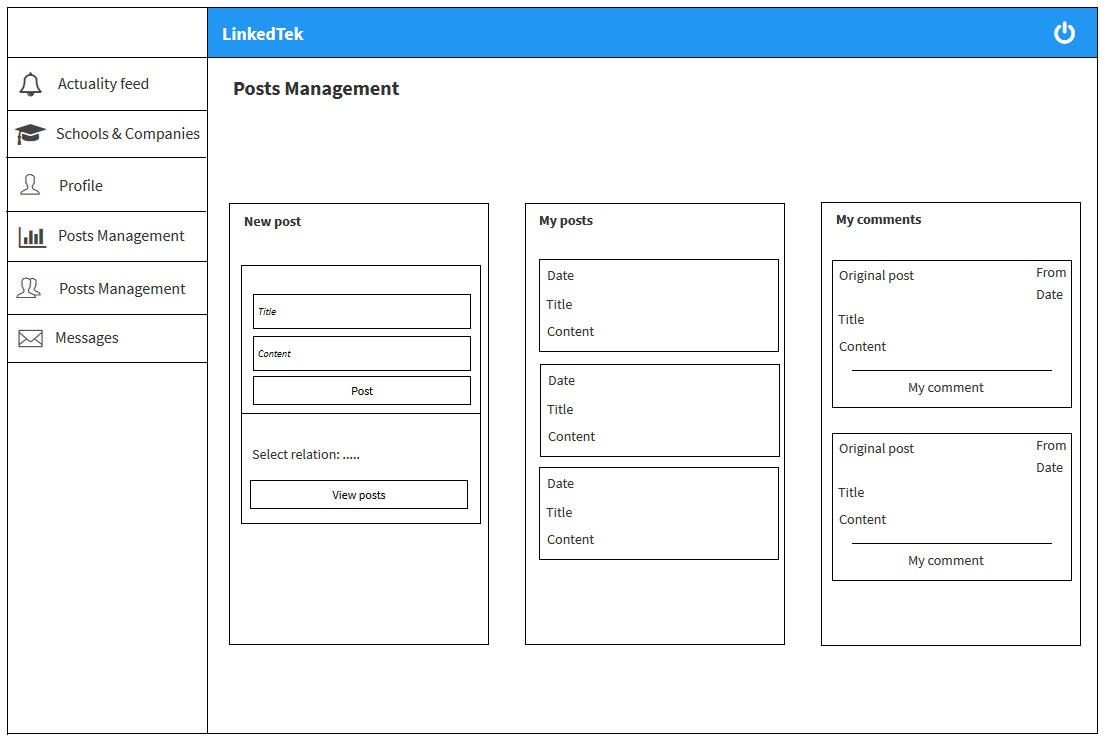


Figure 14 : Frontend posts management mockup

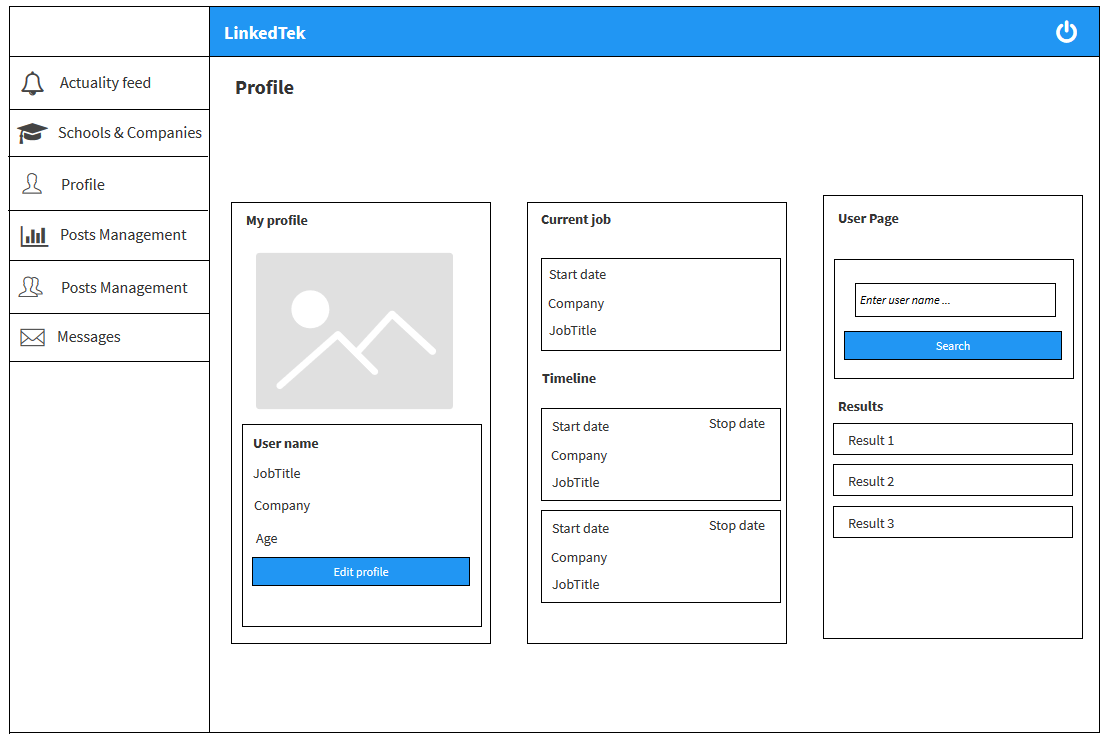


Figure 15 : Frontend user profile mockup

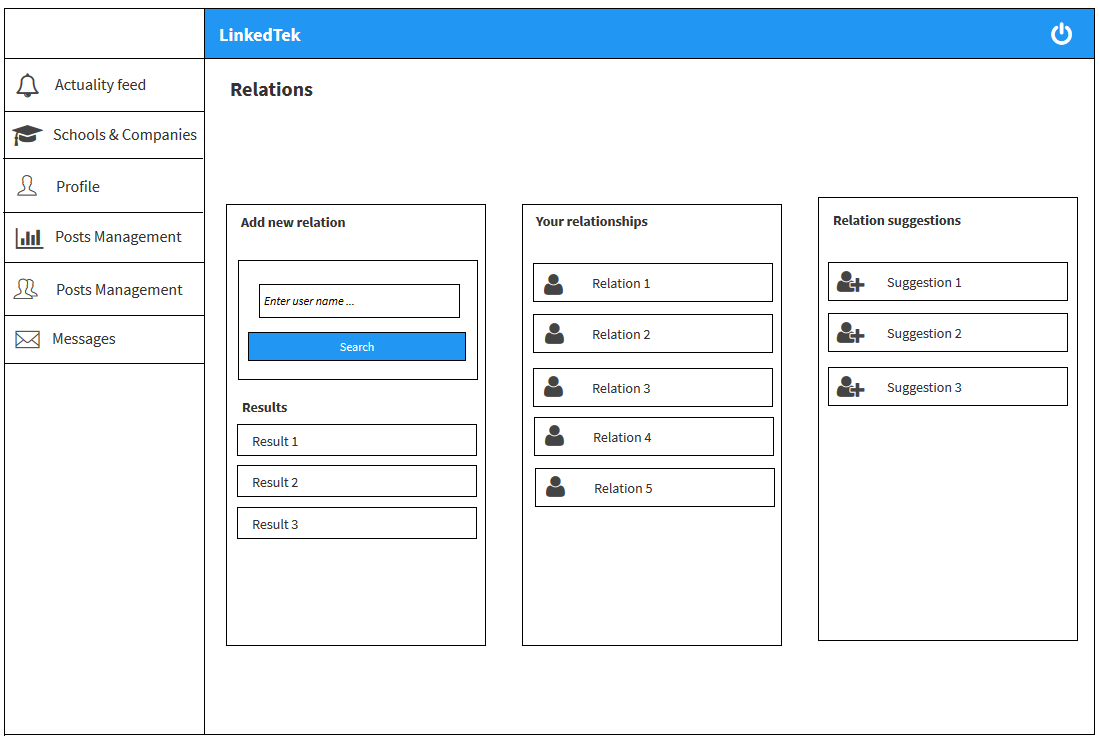


Figure 16 : Frontend relations mockup

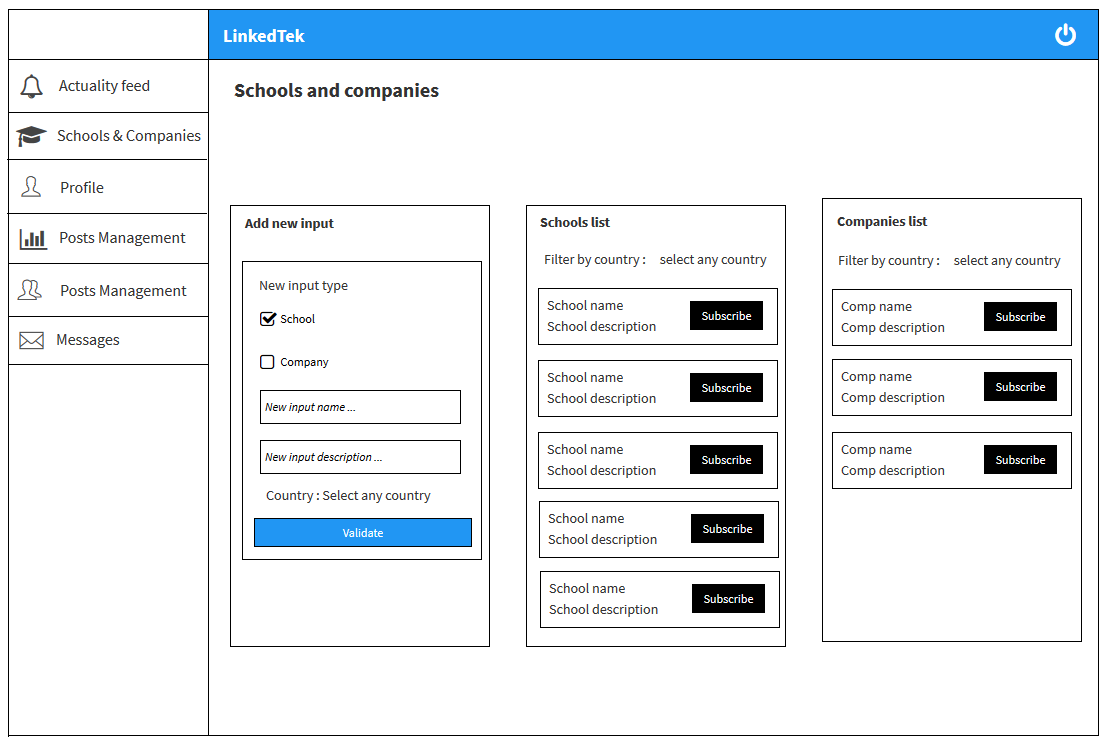


Figure 17 : Frontend schools and companies mockup

# Backend architecture

### Component 1

Description of component 1: utility, main classes, key functions and actions, link to other components

### Component 2

Description of component 1: utility, main classes, key functions and actions, link to other components.

### Component 3

Description of component 1: utility, main classes, key functions and actions, link to other components.

# 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\_FUNC\_010 | You must be able to register / log-in on the software | Frontend:  *Login.js*  *Register.js*  Backend: | Frontend:  *clickOnRegisterButton()*  *clickOnSubmitButton()*  Backend: |
| REQ\_FUNC\_020 | You must be able to add / edit / list the schools | Frontend:  *Schools and companies.js*  Backend: | Frontend:  *componentWillMount()*  *addNewSchoolOrCompany()*  *handleSchoolOrCompanySubscription()*  *handleEditCompanyModalCloseValidated()*  *handleEditSchoolModalCloseValidated* ()  Backend: |
| REQ\_FUNC\_030 | You must be able to add / edit / list the companies | Frontend:  *Schools and companies.js*  Backend: | Frontend:  *componentWillMount()*  *addNewSchoolOrCompany()*  *handleSchoolOrCompanySubscription()*  *handleEditCompanyModalCloseValidated()*  *handleEditSchoolModalCloseValidated* ()  Backend: |
| REQ\_FUNC\_040 | As administrator, you must be able to list all the users | Frontend:  Backend: | Frontend:  Backend: |
| REQ\_FUNC\_050 | As administrator, you must be able to edit / remove /add/ ban the users | Frontend:  Backend: | Frontend:  Backend: |
| REQ\_FUNC\_060 | As administrator, you must be able to delete schools or companies | Frontend:  Backend: | Frontend:  Backend: |
| REQ\_FUNC\_070 | As administrator, you must be able to edit / remove / add all the posts and user’s comments. | Frontend:  Backend: | 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: |
| REQ\_FUNC\_090 | A user can subscribe or unsubscribe in several companies or in several schools | Frontend:  *Schools and companies.js*  Backend: | Frontend:  *handleSchoolOrCompanySubscription()*  Backend: |
| REQ\_FUNC\_100 | A user can see other users’ profiles (schools, companies, ...) | Frontend:  *Relations.js*  Backend: | Frontend:  *handleRelationModalCloseValidated()*  Backend: |
| REQ\_FUNC\_110 | A user can add or remove another user to is relations | Frontend:  *Relations.js*  Backend: | Frontend:  *handleRemoveRelation()*  Backend: |
| REQ\_FUNC\_120 | A user has access to a feed with all is relations’ activities (new post, comment, ...) | Frontend:  *Actualities.js*  Backend: | Frontend:  *componentWillMount()*  Backend: |
| REQ\_FUNC\_130 | A user can add a new post, edit or remove it | Frontend:  *Posts.js*  Backend: | Frontend:  *componentWillMount()*  *handleRemovePost()*  *handleEditPost()*  *handleNewPost()*  Backend: |
| REQ\_FUNC\_140 | A user can comment a post (and edit / remove the comment as well) | Frontend:  *Posts.js*  Backend: | Frontend:  *componentWillMount()*  *handleEditComment()*  *handleRemoveComment()*  *addNewComment()*  Backend: |
| REQ\_FUNC\_150 | The platform needs to suggest to the user new relation (thanks to common markers like schools, companies or relations) | Frontend:  *Relations.js*  Backend: | Frontend:  *componentWillMount()*  Backend: |
| REQ\_FUNC\_160 | A user can send, see and respond to messages from another user. | Frontend:  *Message.js*  Backend: | Frontend:  *componentWillMount()*  *sendNewMessage()*  *handleSendReplyMessage()*  Backend: |

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