# Introduction

This document specifies the System Requirements for the TalkBack system, which is a web-service based messaging system, which enables users to chat with each other and / or play backgammon against each other (or against the computer in future versions).

## Functional Requirements

TalkBack is a messaging system which provides users with the ability to chat with each other and/or play backgammon against each other. The system is comprised of a desktop client application as well as a web browser client application, both of which connect to a cluster of central application servers, this allows users to see other users currently online, and to choose whom to talk to or play backgammon against.

# Presentation layer

This section specifies the functionality of the GUI layer for both client applications by describing their functionality. Detailed GUI layout is specified below in section ‎6.

## Desktop application GUI

Similar to most other desktop applications, the applications’ main window will have a navigation bar at the top, a small footer section at the bottom and a main content area at the middle.

### Navigation bar

The navigation bar of the application will contain a logo / title and a dropdown menu toggle button. The menu functions will be described in section ‎‎6.2.1. In future versions, a “profile settings icon will navigate to a dedicated window.

### Footer

The applications’ footer will be a small display strip at the bottom of the window containing copyright information and app version.

### Login page

If the user is not logged in to the application, the only available page will be the login window. Much like any other login form, this will comprise of a welcoming message reading “Welcome to TalkBack” and in smaller print: “please login or signup to commence using the application”.

Beneath would be a standard login or signup form, containing a “username” text input and a “password” field. The signup form will contain an additional “confirm password” field. Finally, a “submit” button to send the relevant request to the server.

If a user has been previously logged in from this application instance, the login form will be displayed with the “username” field prefilled with the suggested username, and a link to the signup form for new users will show at thew top.

If no previous user is known to have been logged in to this application instance, the signup form will be displayed with a link to login form for existing user showing at the top.

### Main page

Once a user is logged in, the main page will appear.

This window will be divided into two sections:

At the top half of the page, a list of users that are currently online, and at the bottom half of the page, those users not currently online will be displayed. In future versions, both user lists will be limited to current users’ friends, and a separate “find users” page will handle the creation of friend groups and the location of other users.

Additionally, a “chat” button and a “play backgammon” button will be displayed in a prominent location at the top half of the page. The functionality and state of which will be described separately. In future versions, the “play backgammon” button will be changed to allow other games as well.

#### Initiating a chat

There are two ways to actively initiate a chat with another user and one passive way:

* Double-clicking (tapping) a user from the “online users” list
* Selecting (highlighting) a user from that list and then clicking (tapping) the “chat” button.

In both of these scenarios a separate chat window will open with an empty “chat history” pane.

* A “chat request” was pushed from the server with a sender and message body.

In this case, the application will first verify that no existing “chat window” with the specific sender is open. Then a new “chat window” will open with the message body in the chat history pane. If a “chat window” with the specific sender is already open, the message body will be added to the chat history pane of the existing window.

In both cases a “notification” will be triggered, comprised of an audible sound, a visible icon change (see section ‎6.1.3), and bringing the “chat window” to the front (unless user is currently engaged in an active game).

#### Initiating a game

To initiate a backgammon game with another user, that user must be selected (highlighted) from the “online users” list and then the “play backgammon” button clicked (tapped).

When this happens, the other user will receive a “game request”.

As a “game request” is received by a user, a confirmation message will popup informing the receiving user that the other user wishes to play with him and asking whether the user accepts.

If the user accepts, the game window will be launched and an approval message will be sent to the initializing user, in turn launching the game window there.

In future versions, clicking (tapping) the “play backgammon” button without selecting (highlighting) a user will look for a similar request by another user for 15 seconds. If such a request was found, a game will be initiated between those two users. If no such request was found within the allocated 15 seconds, a new game against the computer will be launched locally, displaying a message: “You are playing against the computer”.

This will be achieved by having a cached queue of pending game requests, where a new “random game” request will look for pending requests and if none are found will be added for up to 15 seconds. When a game is initiated with a random user, or when the allocated 15 seconds have passed, that request will be removed from the queue.

### Chat window

The chat window will be a separate window, not a part of the main window. The name of the other user will be displayed in the window title. The top half of the window will show the chats’ history, every message will be displayed in its’ own rectangle with a timestamp at the bottom indicating the time that message was received at its’ recipients’ application. Messages from the different users will be differentiated by the background color and / or pinning to opposite sides of the window (see dedicated design section).

In the bottom half of the window, a text editing area will be displayed. This is where the user composes the next message to send. Clicking (tapping) a “send” button or typing “enter” will clear the text editing area, send a “chat request” containing the message body, and display that message in the chat history pane without a timestamp. Only when an acceptance response is received from the server, the timestamp of the message reception will be added. All messages should appear - in the order they were sent - in the chat history pane.

Somewhere on the window, a “play backgammon” button will be displayed. Clicking (tapping) that button will behave similar to the “play backgammon” button on the main page after selecting the user this chat window refers to.

Currently, no repository is required. When a user closes the chat window, all of the chat history will be lost. In future versions, chat history will be saved for a predetermined period of time, so that when a new chat window is opened with a user that had previous chats with the current user, the old messages will be displayed.

### Game window

The game window will be a separate window, not a part of the main window. The name of the other user (opponent) will be displayed in the window title, in future versions, in case of a game against the computer, the title should say “Computer”. The game window will be split into two main areas: a full backgammon board on one side, and an action pane including a “chat” button, a “resign” button, and the game controls.

The rules of backgammon are known, and a tutorial will be found in the menu of the main window, under: Help > How to play Backgammon.

At the start of a game, both dice will be rolled, each representing one user (right die representing current user, left die representing opponent). The user with the higher number goes first, in case of a tie, the dice will be rolled again (with a short delay so the users will understand what happened, see dedicated section).

By default, every user will see themselves as white and the opponent as black, their home pane will be placed at the bottom right of the board. Optional extension will allow changing these settings, this will only affect the display, and have no impact on the rest of the game logic.

At the start of a users’ turn, both dice will roll, displaying the results simultaneously for both users. The game controls will be enabled, and a timer will start counting down two minutes. From the moment each users’ turn begins, they have 2 minutes to make their moves. If they don't move within this time, they resign from the game and the opponent is declared winner. In that case, a message will pop up, notifying the user they haven't made a move in time and lost the game.

The user whos’ turn it is will then be able to select chips to move by clicking (tapping) the origin field. This will “pickup” a chip from that field (see visual effects section). (A user will not be able to “pick up” opponents’ chips).

If a chip is captured, at the beginning of the users’ turn, it will automatically be “picked up. The user cannot put down a captured chip. If no possible legal moves are available, a message saying “No possible moves” will blink simultaneously on both users’ boards before returning the turn to the opponent.

When clicking (tapping) a legal drop location or a field with a legal drop location the chip will appear in the appropriate field (see visual effects section).

A “picked up” chip can also be put back down in its’ original location (unless it is a captured chip), allowing the user to select a different one.

After one chip was moved, as long as there is a potential move remaining for the current turn, an “undo” button will be enabled, reverting the last move.

All this logic will only be displayed on the current users’ game board, until the last move is completed (to avoid displaying moves that will be undone later). During the opponents’ turn, all the game controls will remain disabled.

Once the last chip was dropped at its’ location, The current users turn is over. All the game controls will become disabled, and a “game move” request will be sent to the opponent.

When a user receives a “game move” request, a short animation will display the opponents’ moves during the last turn (see visual effects section), and the current users’ turn will start.

When a user clicks (taps) the “resign” button, a confirmation message will pop up asking whether the user is sure about resigning the match. If confirmed, the game will end as a loss.

When a user finishes removing all of their chips, the game is won, a “game move” request will be sent to the opponent and a message saying “Congratulations, you won” will be displayed. Meanwhile at the opponents’ side, the last move will be displayed and a message saying “You lost, better luck next time” will be displayed. Upon closing that message, the game window will be closed.

### Taskbar icon

Upon launching the application, the client window will open, at the same time the applications’ icon will appear in the lower right-hand corner of the taskbar in the operating system (most commonly Microsoft Windows), as is with other messaging programs.

The appearance of the icon will show the user is offline, until the user logs in to the server. Then the icon will change as described in section ‎6.1.

Mouse left-clicks on the icon should preserve default functionality, right-click will have custom functionality as detailed in section ‎6.1.4.

### Notifications

When a user receives a new chat request, a new game request, or a new game move request, an audible sound will alert the user as well as a notification mark that will appear on the taskbar icon (see section ‎6.1.3).

## Web client application GUI

In addition to the desktop application described above, a web client application will allow users to connect to the same services and perform similar actions via a web browser.

For all intense and purposes, the web client should be similar to the desktop client, apart for the following:

### Remembering a previous user

Most browsers remember form data by default. No custom cashing system will be necessary. The login form will be displayed by default, with a link to the signup form from new users. Other than the basic session data and authentication token, no data will be cached on the clients’ side.

### Application icon and notifications

The custom appearance and behavior of the desktops’ application taskbar icon will be mirrored in the web applications’ tab icon (see section ‎6.1). Notifications will utilize the available browsers notification options without requiring any custom permissions on the clients’ machine.

### Opening additional windows

When actively initiating a chat or a game, the new chat or game window can be implemented as a custom popup prompt, allowing several chats or games to be displayed side by side simultaneously. Or the option to open in new tab or new window will be available, depending on the size of the display port. When a chat or game request is received, and should open a new window, a popup message will prompt the user to allow this.

# Application layer

This section will describe some of the application components and the communication between them.

## Authentication service

A web API service to register users, and handle login and logout requests. This service requires a small database to store user login information such as username and password. In the future, potentially last login, last password change, two factor authentication data extra. Additionally, any data required to handle a JWT token generation, renewal, and expiration.

### Register request

When a new user wants to register (signup) to the application, a registration request will be sent to this service. This request needs to contain a unique username comprised of an alphanumeric string, between 5 and 15 characters, with no spaces or special characters. The username needs to be a case insensitive identifier, no two users can have the same username. The request should also contain a password, adhering to standard password rules. In future versions, additional data such as two factor authentication data, initiating IP address, and security related data might be added to the request.

After confirming that no user with the same username exists in the database (case insensitive), and the password is valid, the service will create the new user record in the database, and automatically login that user.

The password will be hashed and saved only in its’ encoded form; no password will be saved as plain text.

In case of an existing username, the response should reflect the problem, and in the future, include suggestions for similar available usernames.

### Login request

Similar to the registration request, this request will contain the same fields. Handling this request will include finding the relevant user in the database and comparing the hashed password to the value stored in the database. In future versions, additional security checks will be performed such as TFA, IP verification ext.

If all is found to be satisfactory, a response will be sent to the user including a JWT token with a short (5 – 30 minutes) expiration time and a refresh token saved in the appropriate fashion.

When a user sends a request to any of the applications services, that request has to contain both the JWT access token as well as a valid refresh token. On the server side, first the JWT access token will be validated. If the JWT token is valid and not expired the request will be handled normally. If the token is expired, a refresh token request will be sent automatically, and only upon successful completion of the refresh token procedure, the original request will be handled.

### Refresh token request

This request will be sent automatically when a valid yet expired JWT token is detected. It will contain the username and the refresh token provided by the user. This data will then be verified against a similar key-value pair stored in the authentication service, and if found valid, a new JWT token with new expiration time will be returned. If one of these steps fails, the user will be required to login again.

### Logout request

When a user wishes to logout, a logout request containing only the username will be sent. The authentication service in turn will remove the username / refresh-token pair from the internal storage so no further refresh token requests will be answered. In future versions, additional data might be required.

### JWT token

This application will use the JWT stateless authentication method. Using bearer token in every request header to identify and verify the user. So far, only the username has to be present in the “claims” of the token. The token will also have a short (5 – 30 min) expiration time.

### Refresh token

This token will be generated at the beginning of a users’ session, upon login. It will be stored internally with the username as a key-value pair in the authentication service, and it can theoretically be a standard Guid (although for future advanced functionality a JWT token with a long expiration time is preferrable).

## Online users service

This service will keep track of users who are online in real time, notifying all users when another user has joined (logged in) or left (logged out or dropped connection).

### Notify online request

When a user has successfully logged in to the application, this service will be notified. This can be achieved by one of two ways:

* The client applications will automatically send a notification after a successful login or logout. This means having every client application responsible for registering the user as online or offline. But will also allow control over this function if in the future some new feature will require a user to login without being announced to all other users.
* The authentication service will notify the chat service of any new login or logout requests. This will eliminate the need for client applications to register after a successful login or before attempting to logout. But it is more complicated to implement and more ridged in regard to possible future changes.

### Notify online push request

After a user has logged in and notified the service of being online, the service should push a “notify online push request” to all users that are currently online, notifying them of the new user. This will allow the client application of all users to adjust their display of that user.

### Notify offline request

### Notify offline push request

## Chat service

This service will also have to keep track of users who are online in real time in order to allow users to converse with one another.

This can be achieved in several ways:

* Every client application will be responsible to register to the chat service after logging in and registering to the “online users service”. This is a simple solution to implement, but it requires separate steps to be taken by the client application, as well as duplicating a lot of the “online users service” logic.
* The “online users service” will forward the registration request from the client application to the chat service. This will eliminate the need for another separate registration step on the clients’ side, but will still duplicate some logic of the “online users service” such as keeping a live collection of registered users and a map of usernames to socket ids. From the simplicity standpoint, it is a nice compromise.
* The chat service will not handle any socket related logic. The “online users service” will expose a public endpoint “send message to user” that will receive the recipients’ username and a generic JSON object message body and push it to the relevant user. In this solution, the chat service will only serve as a proxy, receiving messages and forwarding them through the “online users service”. The client application of the receiving user will have to differentiate between the message body types to handle them correctly.

### Chat request

When a user wishes to initiate a chat, the client application will send a “chat request” to this service. The sending user will be identified by the authentication token, and the chat request will contain the requested recipient as well as the message body.

### Chat push request

If the recipient is online, the service will push a “chat push request” to that user, awaiting a confirmation response from the recipient.

Upon receiving the confirmation from the message recipient, the service returns a “message received confirmation” to the sending user. This will allow the client application of the sending user to display a timestamp at the bottom of the message pane, indicating it was successfully sent and received (see dedicated section).

In case any one of the steps mentioned above encounters an error, the sending user will be notified, and the client application will display a disclaimer at the bottom of the message pane indicating it wasn’t received (see dedicated section).

### Chat history

Currently, saving previous messages on the server side is not required. Therefore, no database will be necessary for this service. In the future, however, some type of repository might be required.

## Game service

Like the chat service, this service will also keep track of online users in real time. This will let users engage in a game of backgammon against each other (see section ‎3.3).

### Initiate game request

The first stage of initiating a game is the first game proposal from a user. The sending user will be identified by the authentication token, and the request will contain the username of the requested opponent as an optional field, so in future versions a “random opponent” game request will be possible.

### Initiate game push request

When a user has sent an “initiate game request” the service will find the requested opponent in the “online users”. If the opponent is online, the service will push a “initiate game push request” to the opponent.

Upon receiving the “initiate game push request” a confirmation message will pop up on the clients’ side, awaiting a “confirm” or “reject” choice by the user for one minute. If the user chooses to accept the game proposal, the service will notify the proposing user and at the same time both client applications will open a new game window and start the game session.

In case any of the above steps are not completed, whether the opponent is not online, the push request failed to send, the opponent chose to reject the game proposal, or the opponent didn’t react within the allotted time – an appropriate message will be shown to the initiating user and all popups will close.

### Notify dice request

When a users’ turn starts and the dice are rolled, the opponent will immediately be notified of the dice results. This will happen with one die at the start of the game, to determine which user gets to go first, as well as the beginning of every turn, even if the resulting numbers are illegal moves (in this case both users will see a message declaring “no possible moves” and the turn will end). If only part of the moves are legal, the dice representing the illegal numbers will show as disabled or used, see presentation section for examples.

### Notify move request

When a user has moved their chips, at the end of their turn, the opponent will be notified of the users’ moves so they can be displayed on the board (see presentation section). This will coincide with the end of one users’ turn and the beginning of the other users’ turn.

### Resignation notification request

When a user chooses to resign the mach, or a user has not finished their turn within the allotted time, the opponent will be notified of the resignation and the end of the game.

### Game history

In the current version of the application, no part of the game, moves or results, should be saved. This means that no database is required, and no “game entities” are necessary. But in future versions, there might be a grading or points system for game statistics as well as saving a game to be continued or replayed in the future.

### Initiate random game request

This feature will only be available in future versions. This is the same request as the “initiate game request” without a requested opponent. When this request is received, a short-term cache will be checked for any similar pending requests. If such a request is found, both of the requesting users will be matched with one another, and a game will be initiated between them. If no such request is found, this request will be added to the short-term cache for 15 seconds, awaiting other requests. If no such request is found within the allotted time, the request will be refused and a game against the computer will start on the requesting clients’ application.

# System architecture

This section describes the applications’ general architecture from the technical point of view.

## Server architecture

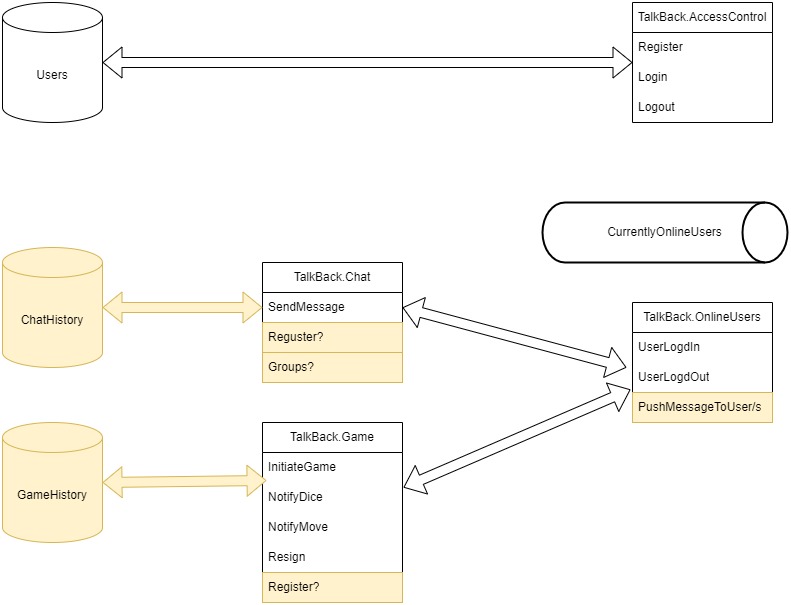
The server side of the application will be implemented using the “microservices” approach. This means that there will be several server applications, each standing alone as an independent web service, communicating with each other through http / tcp requests and possibly a socket or hub connection. Each service could technically be implemented in a different programming language (C#, C++, javascript, typescript, python, php, ext…) on a different platform (ASP.NET Core, node.js, web2py, ext…) and installed on a different machine. The services that need to communicate with each other will have a configuration file containing the other services’ urls so the value will be editable without redeployment of the code.

The services will be separated as described in the application layer section (see section ‎3).

Any service requiring a repository (database), will have its’ own small database of an appropriate type (SQL, mongo, blob, ext…). common entity keys will be used to cross reference the data from different databases.

In the future there is a possibility of using docker to manage the different parts of the application’ server for better administration capabilities and added security.

### Server architecture diagram:



## Client architecture

The client-side applications can also be implemented using any programing language or framework.

The desktop application could be a WPF application using .NET Core or a React or Angular application using Electron or any other desktop application technology as long as it will run on any standard desktop or laptop computer. Regardless of the language and framework, the appropriate design standards will be applied such as MVVM for WPF or MVC for angular.

For the web application a single page approach will be preferred, and clean, readable, design patterns expected.

It is important to emphasize that each client-side application will contain the exact same functionality and consistent behavior. The user will choose one application to work with and it will allow all of the requested functionality except the taskbar icon or popup behavior that will be adjusted to fit the frameworks capabilities (see section ‎2.2).

# Nonfunctional requirements

This section describes overall application requirements that are not covered under the previous sections.

## Data management

Currently, the only data that must persist when the server goes offline is the authentication user base. This data can be stored in a single table, containing the username (unique, possible primary key), and the hashed password (don’t save the password as plain text). Any additional data is optional, and any more sophisticated password encoding is not required at this point.

In the future, when other services will require data persistence, the primary key of the users table can be used as a reference.

Any type of database is acceptable, as long as it fulfils standard security requirements (unlike a text file stored somewhere on a public file system). Connection strings should be configurable and connection credentials encoded.

## Performance

Any request originating at the client application and awaiting a response from the server should be completed within less than 3 seconds, unless extreme external conditions are interfering with the connectivity. This should suffice for the desktop client application as well as the web client application.

## Security

As mentioned above (see section ‎5.1), passwords should not be saved as plain text, for both the user database as well as the connection strings.

The JWT token should expire after 5 minutes requiring a refresh token. This refresh process will be invisible to the user, happening in the background of the client and server applications as long as the user didn’t log out. Every attempt to access any one of the services must be verified and authenticated using the JWT token.

# Presentation and UI effects

This section describes the UI components and proposes some visual effects for better user experience.

## Taskbar / tab icon

The following are some suggestions as to how the client applications’ icon should look and how it will indicate application status.

### Offline icon

One way to indicate that the user is offline or disconnected is by showing a gray icon:



### Online icon

In contrast, when the user is online and logged in the icon should be colorful:



### Message alert icon

When the user received a new chat or game message the icon should reflect it:



### Taskbar icon functionality

Other than indicating the applications’ status, the taskbar icon should have some built-in custom right click functionality.

|  |  |
| --- | --- |
| Option | Functionality |
| Sign-in | Enabled when there is no user currently logged in to client application.  When selected the login window will popup and user will be able to login. |
| Sign-out | Enabled when a user is logged in to client application.  When selected the user will logout. |
| Open | Always enabled.  When selected brings the contacts window to front of screen |
| Exit | Always enabled.  When selected the application will close, if a user is logged in, it will logout for him. |

## Navigation bar

This section will describe the navigation bar UI. The color scheme and exact design is only for demonstration purposes and can be adjusted to fit the general applications’ design.



|  |  |  |
| --- | --- | --- |
| Component | Description | Image |
| Title | The name of the application, text or logo. Tapping this title will navigate back to home (contacts) page. |  |
| Greeting | A greeting with the username of the current user. |  |
| Drop down menu button | This button will toggle the drop-down menu. The red dot at the top left corner indicates that a new message or game request was received. |  |
| Profile button | In future versions, tapping this button will navigate to a “profile edit” page. |  |

### Drop-down menu actions

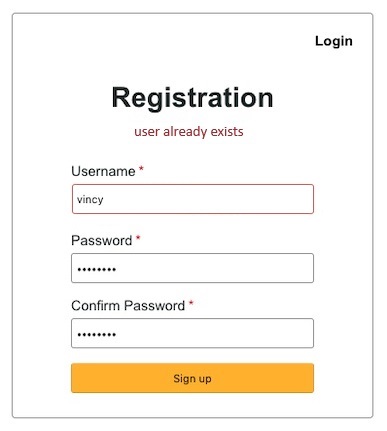
Tapping the drop-down toggle button will show a menu containing the fallowing actions:

|  |  |
| --- | --- |
| Option | Functionality |
| Connect | Enabled when there is no user currently logged in to client application.  When selected the login window will pop up and user will be able to login. |
| Disconnect | Enabled when a user is logged in to client application.  When selected the user will logout. |
| Actions | Enabled when a user is logged in to client application.  Containing two sub menus:  **Chat** – Enabled when a user is selected (highlighted), does the same as tapping the “chat” button.  **Play** - Enabled when a user is selected (highlighted), does the same as tapping the “play” button. |
| Help | Always enabled.  Containing two sub menus:  **How to Play** - launches a tutorial on how to play backgammon.  **About** - shows info about the program. |

## Register / Login form

This section will describe the registration and the login form UI. The color scheme and exact design is only for demonstration purposes and can be adjusted to fit the general applications’ design.

### Register form



The registration form consists of the fallowing components:

|  |  |  |
| --- | --- | --- |
| Component | Description | Image |
| Link to Login-form | Tapping this link will replace the registration form with the login form |  |
| Title | The forms’ title, exact text might be revised. |  |
| Error display pane | When a validation error occurs, or the registration process fails the relevant error text will be displayed here. |  |
| Username-field-label | The label marking the username field, the labels’ text might be revised. The red dot to the right of the text is to indicate the field is mandatory. |  |
| Username-input-field | The input field where the user inserts their desired username. The username will have to be unique. |  |
| Password-field-label | The label marking the password field, the labels’ text might be revised. The red dot to the right of the text is to indicate the field is mandatory. |  |
| Password-input-field | The input field where the user inserts their desired password. The password will have to be of a minimum complexity (Letters, numbers, special characters…). |  |
| Confirm-Password-field-label | The label marking the confirm password field, the labels’ text might be revised. The red dot to the right of the text is to indicate the field is mandatory. |  |
| Confirm-Password-input-field | The input field where the user inserts their password again. The value will have to be the same as the password field. |  |
| Submit button | Tapping this button will submit the registration data. The exact text might be revised. This button will only be enabled when all the form fields are valid. |  |

### Login form

A login screen with a yellow box and black text

Description automatically generated

The registration form consists of the fallowing components:

|  |  |  |
| --- | --- | --- |
| Component | Description | Image |
| Link to Register-form | Tapping this link will replace the login form with the registration form |  |
| Title | The forms’ title, exact text might be revised. |  |
| Error display pane | When a validation error occurs, or the login process fails the relevant error text will be displayed here. |  |
| Username-field-label | The label marking the username field, the labels’ text might be revised. The red dot to the right of the text is to indicate the field is mandatory and the “required” error message will appear when an attempt to submit the form without the username data occurred. |  |
| Username-input-field | The input field where the user inserts their desired username. The username will have to be unique. |  |
| Password-field-label | The label marking the password field, the labels’ text might be revised. The red dot to the right of the text is to indicate the field is mandatory. |  |
| Password-input-field | The input field where the user inserts their desired password. The password will have to be of a minimum complexity (Letters, numbers, special characters…). |  |
| Submit button | Tapping this button will submit the registration data. The exact text might be revised. This button will only be enabled when all the form fields are valid. |  |

## Contacts page