# **Running and Setting Up the Gamification Application**

## **Prerequisites**

Make sure you have the following prerequisites installed on your machine:

* [.NET SDK](https://dotnet.microsoft.com/download) (version 7.0 or later)
* A code editor such as [Visual Studio Code](https://code.visualstudio.com/) or [Visual Studio](https://visualstudio.microsoft.com/)

## **Clone the Repository**

Clone the Gamification application repository from the source:

git clone <repository-url>

cd Som\_Gamification.UI

Replace <repository-url> with the actual URL of your Git repository.

## **Running the Application**

## Open a terminal and navigate to the project directory:

cd Som\_Gamification.UI

Run the application using the following command:

dotnet watch

This command builds and runs the application, enabling hot reload for faster development. Hot reload allows you to see the changes in real-time without restarting the application.

Once the application is running, you should see output similar to the following:

dotnet watch 🔥 Hot reload enabled. For a list of supported edits, see https://aka.ms/dotnet/hot-reload.

💡 Press "Ctrl + R" to restart.

dotnet watch 🔧 Building...

Determining projects to restore...

All projects are up-to-date for restore.

Gamification.UI -> /path/to/Som\_Gamification.UI/bin/Debug/net7.0/Gamification.UI.dll

dotnet watch 🚀 Started

info: Microsoft.Hosting.Lifetime[14]

Now listening on: https://localhost:5001

info: Microsoft.Hosting.Lifetime[14]

Now listening on: http://localhost:5000

info: Microsoft.Hosting.Lifetime[0]

Application started. Press Ctrl+C to shut down.

info: Microsoft.Hosting.Lifetime[0]

Hosting environment: Development

info: Microsoft.Hosting.Lifetime[0]

Content root path: /path/to/Som\_Gamification.UI

* Access the application in your web browser:
  + [https://localhost:5001](https://localhost:5001/) (for secure connection)
  + [http://localhost:5000](http://localhost:5000/) (for non-secure connection)
* To stop the application, press Ctrl+C in the terminal where the application is running.

**Functional Description**

* Currently, student performance on GBI Case Studies is recorded and displayed in a Fiori app, Global Bike Monitor, that enables the professors to see student performance on each step for a specified Case Study. This functional requirement is to make the information in the GBI monitor available in JSON format through an API to be consumed by another application, Gamified GBI Web App.

**Gamification Logo => Breakage fixed, Redirects to Welcome Page.**

* Index Page

**Dashboard**

**New Change => Case Studies Toolbar**

* The Case Studies Toolbar provides quick access to different SAP case studies. Users can navigate to specific case studies such as Accounts Payable, Accounts Receivable, Materials Management, Production Planning, and Sales and Distribution.

**Congratulations Card**

* The Congratulations Card is displayed when a user completes a certain percentage of a process. It includes the user's name, completion percentage, and encourages users to continue their tasks to earn more points and badges.
* New Change => The text dynamically changes based on the percentage of process/tasks.

**Points, Levels, Medals, and Badges Cards**

* These cards display the user's current points, levels, medals, and badges. Badges are dynamically loaded based on the user's achievements in various SAP processes.

**Growth Chart Card**

* The Growth Chart Card visualizes the user's score and steps over time. It includes a chart displaying points and steps, along with information about the total number of steps.

**Steps Report Card**

* The Steps Report Card, currently commented out, could potentially provide additional details about the user's progress in terms of steps taken.
* New Change => Wording change from “Progress” to “\_\_\_\_\_\_”
* **New Change => Growth Chart/Steps Progress Calculation Logic**
* /\*
* This code is designed to calculate the progress of steps completion based on a set of predefined rules.
* The primary goal is to determine the fulfillment percentage, update the ViewBag accordingly, and extract relevant information.
* Variables:
* - PointsDictionary: A dictionary containing predefined points for specific step keys.
* - fulfill: A string representing the fulfillment percentage, initially set to "0".
* - stepsCompletedDictionary: A dictionary to track whether at least one step has been completed for each StepsCount.
* - dictionaryList: A list containing key-value pairs parsed from a JSON token.
* - point: An integer variable to store the total points.
* - level: An integer variable to store the current level.
* - badge: A string variable to store the badge information.
* - Points: A list to store the points corresponding to each step.
* - Steps: A list to store the keys corresponding to each step.
* Steps:
* 1. Iterate through each key-value pair in the JSON token and create a list of dictionary models.
* 2. Initialize Points and Steps lists to store points and step keys.
* 3. For each item in the list, check if it corresponds to a step. If yes, add the points to the Points list, update the fulfill percentage, and track step completion.
* 4. Update fulfill based on the "FulfillmentAll" key in the JSON token.
* 5. Extract and store Points, Levels, and Badge information.
* 6. Check if at least one step has been completed for each StepsCount. If yes, set fulfill to "100".
* 7. Remove "%" if it exists in the fulfill string and update the ViewBag.Fulfillment.
* 8. Update ViewBag with Point, Levels, Badge, PointsList, StepsList, and StepsCount information.
* Usage:
* - This code is likely part of a larger system, where it calculates and displays the progress of completing steps.
* - The ViewBag is used to pass calculated values to the front end, allowing dynamic rendering of progress, points, levels, and badges.
* Example:
* - Suppose the JSON token contains information about completed steps, total points, current level, and badge.
* - The code processes this information and updates the ViewBag for rendering on the user interface.
* \*/

**Leaderboard**

* Displays player information in cards for each player. The cards include a badge with a medal icon, the player's username, and their earned points. The player cards are dynamically generated based on the data provided by the Model object, which is an IEnumerable of LeaderBoard models.
* The view dynamically generates content based on the data retrieved from the server. It uses a loop to iterate over the leaderboard data and create a card for each player, displaying their username and points.
* New Change => Created Indexing/Serial Number on the Users Card.

**Create Users**

* Allows Admin to create users with the functionality to create either one user or multiple users at a time.
* Default Password, UserName, Name are provided while Admin can always create their own particular info.
* We have 2 parameters startingIndex, aims at starting index for generating data, numberOfUsersToCreate, aims at number of users to create.

**Reset Password**

* Allows Admin/Users to reset their password.
* Whenever a new user is created and tries to login with default password, the reset password view is called.

**Badges**

* Retrieves and displays badges for a specified case study.
* Parameter passed is caseStudy, The case study identifier (default is "MM").
* Result of the operation is a view displaying all the badges.