Advanced ISD project report – Team TriMy

Tristan Bernatzik and Mykola Subtelnyi

Content

General

This documentation aims to describe our web-based memory card game which was built using Flask. Our app includes the following features:

- User authentication with email verification (using SendGrid API)
- A memory card matching game with 8 pairs of cards
- Scoring system based on completion time and number of moves
- User statistics tracking with high scores
- Leaderboard showing top 10 players

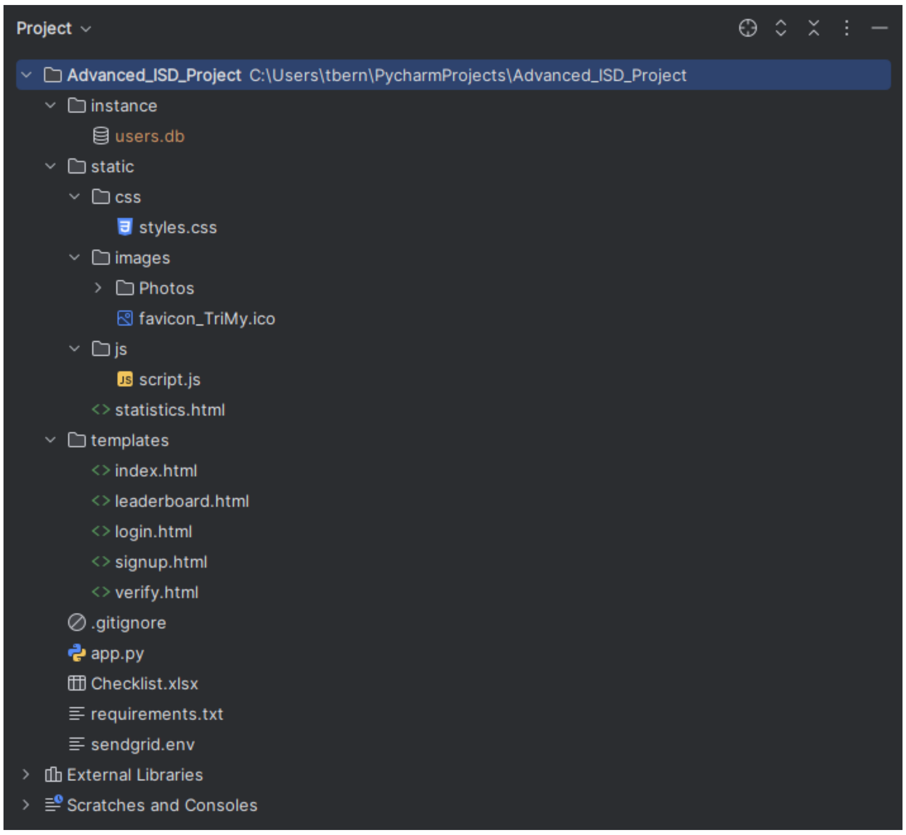
It includes interactive features like card flipping animations, timer, move counter and has responsive design for different screen sizes. Users need to create an account with a Gmail address before being able to play.

The frontend uses HTML, CSS, and JavaScript while the backend is built with Flask, handling user authentication, game logic stores data with SQLite.

Server and client-side interact through AJAX requests for seamless gameplay - when a card is clicked, the frontend sends a request to the backend, which processes game state changes and returns JSON responses that update the UI without page reloads.

File structure

Our file structure looks like this:



```
Project
Advanced_ISD_Project C:\Users\tbern\PycharmProjects\Advanced_ISD_Project
  instance
    users.db
  static
    css
      styles.css
    images
      Photos
      favicon_TriMy.ico
    js
      script.js
    statistics.html
  templates
    index.html
    leaderboard.html
    login.html
    signup.html
    verify.html
  .gitignore
  app.py
  Checklist.xlsx
  requirements.txt
  sendgrid.env
External Libraries
Scratches and Consoles
```

Goal and uniqueness

The goal of this project was to create a memory card game which can be played online by uni.li students. To provide some form of competitiveness the scores of the players should be stored and presented in a leaderboard.

Memory is a round-based card game in which the player needs to find and collect pairs of matching cards.

One turn consists of flipping over two cards and checking the picture on them. Then there are two possible outcomes:

1. The two cards do not match
2. The two cards do match

For the first scenario the number of moves will be increased but both cards are flipped over again. For the second scenario the number of moves will be increased as well but the two cards will stay open, and the number of found pairs is increased by 1. After that the next moves is started.

After all pairs have been collected a score will be calculated based on the number of moves and the time taken. We have decided to use a penalty system where the player starts with 10.000 points and gets points deducted for every move he/she needs more than the minimum (8) and for every second it takes him to finish the game. We decided to penalise wrong moves more than taking more time to make sure the player really thinks about his moves rather than just spamming moves and hoping he/she'll get lucky – even though we are aware that to some extend, luck will always be a factor.

Compared to other memory games already available online our solution is unique that it selects 8 random pictures from a pool of 20 pictures. This makes sure that every game is slightly different, and the player must pay close attention to the pictures every round he/she plays as he/she is not able to just recognise them from earlier round.

Additional uniqueness of this project is that if we can establish, so that only students with uni.li emails can register (unblocking verification codes from the server system), we can do some competition among students in cooperation with Spinnerei (student organisation). Or it can be used at the beginning of every semester as an integration for students.

Contribution

- Tristan
    - Initial app setup + file structure
    - User interface and styles
    - Basic game logic
    - Setup GitHub repository
    - Setup PythonAnywhere
- Mykola
    - Signup procedure set-up
    - SQL database set-up
    - SendGrid setup
    - Leaderboard development
    - Development of multi-mode game (still in development)

We used W3Schools, StackOverflow and AI tools like Gemini, ChatGPT and Claude to get a better understanding of how web development with Flask works. Especially how the integration with JavaScript works to communicate between front- and backend. We also used generative AI for creating chunks of the code, fixing bugs in our code as well as cleaning up some parts of our code as it got messy sometimes.

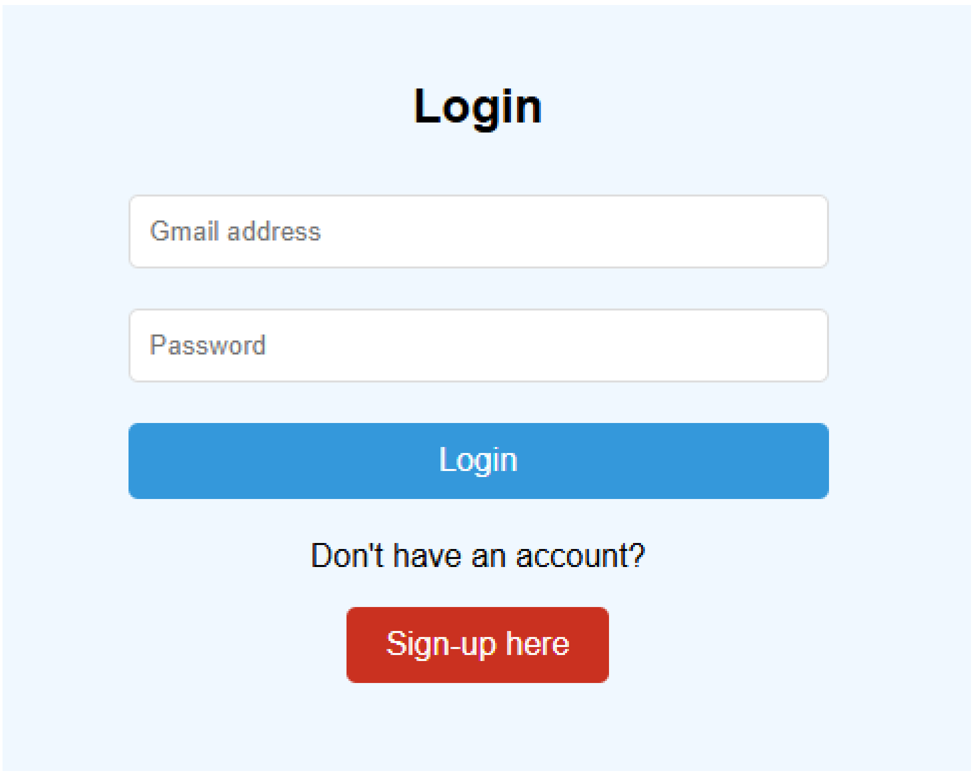
We also used Gemini to help with some Git issues we faced when trying to pull new code to our PythonAnywhere environment as there were some weird merging issues sometimes.

User Guide

In this section we want to provide an instruction for users what they can expect from our website and briefly explain the provided functionalities.

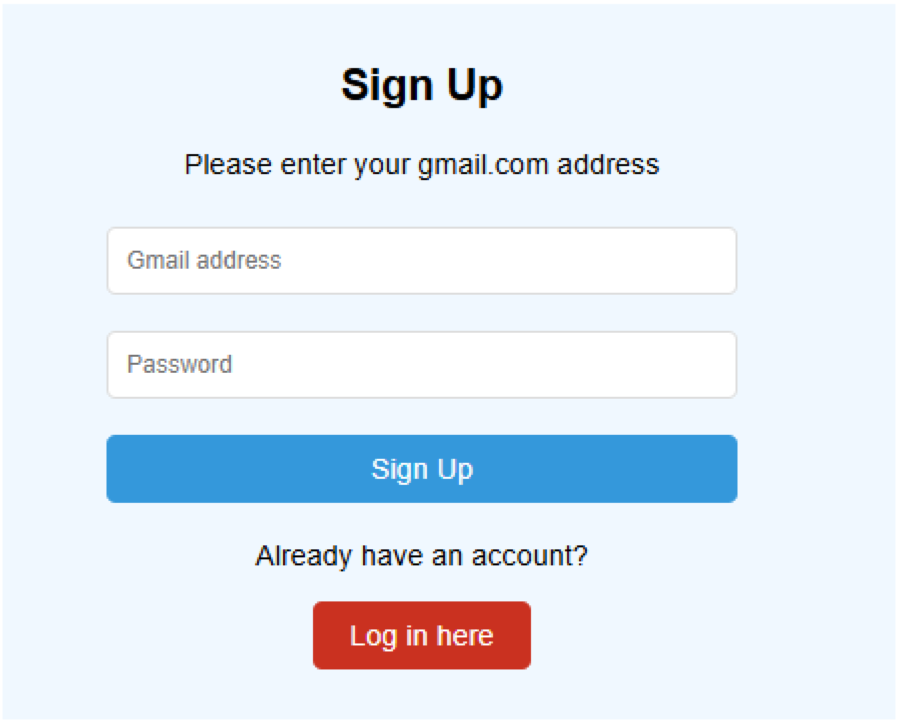
Log in

When a user visits our website, he/she will find the log in screen where he/she is asked to enter with a gmail.com address. If he/she has not registered himself yet he/she can navigate to the sign-up screen by clicking the “Sign up here” button. If he/she has already done this step, he/she can simply continue with the login by entering his username, which is his gmail.com email address, and the password which he/she set during the registration process.



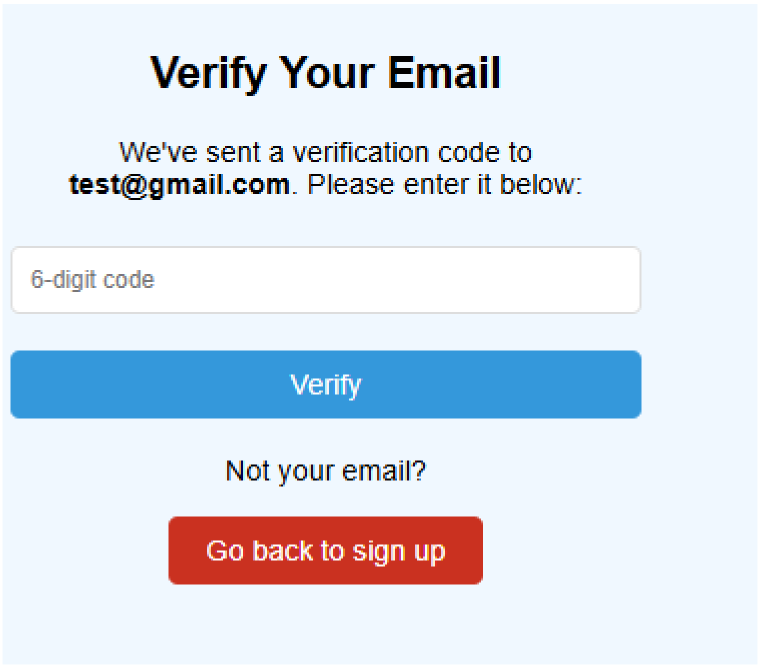
Sign-up

To create his account the user needs to enter his @gmail.com address and his wished password. After he/she clicks on the “Sign up” button he/she is redirected to the Verify page.



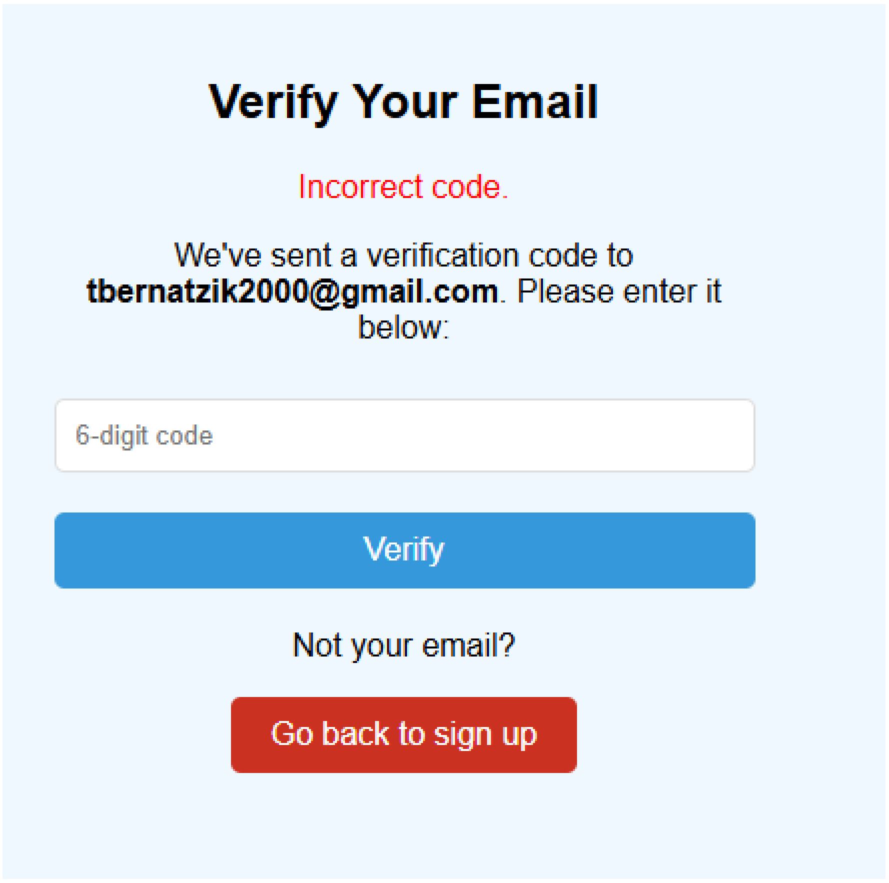
Verify

In this screen the user needs to paste the verification code which he/she received in his/her mail.


Verify Your Email
We've sent a verification code to **test@gmail.com**. Please enter it below:
6-digit code
Verify
Not your email?
Go back to sign up

Once he/she clicks on the “Verify” button and the code is correct he/she is redirected to the login page where he/she can continue the login process.

In case the code is not correct a small message is displayed:


Verify Your Email

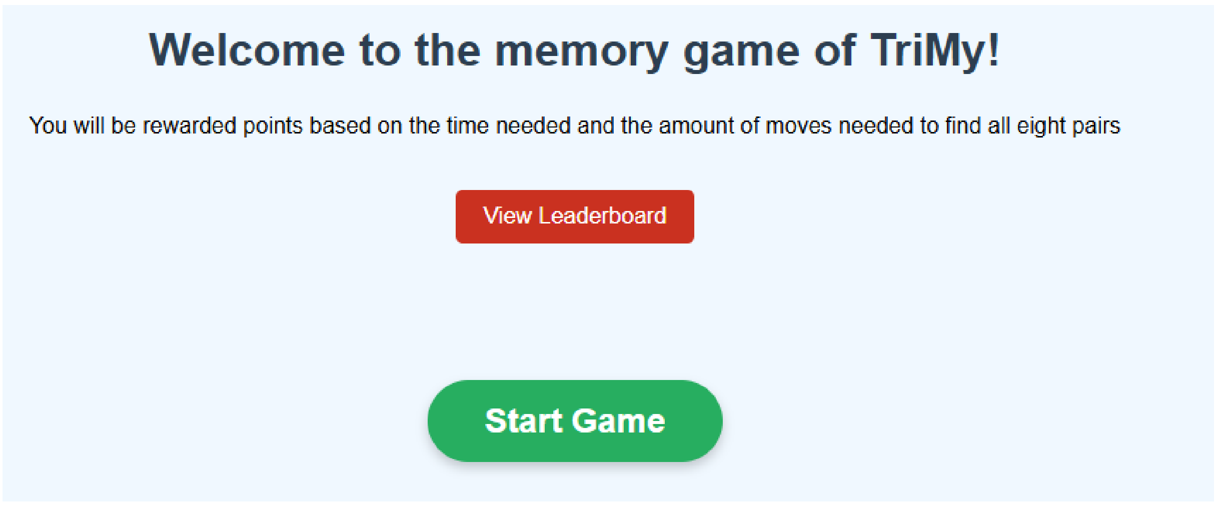
Incorrect code.

We've sent a verification code to **tbernatzik2000@gmail.com**. Please enter it below:

Not your email?

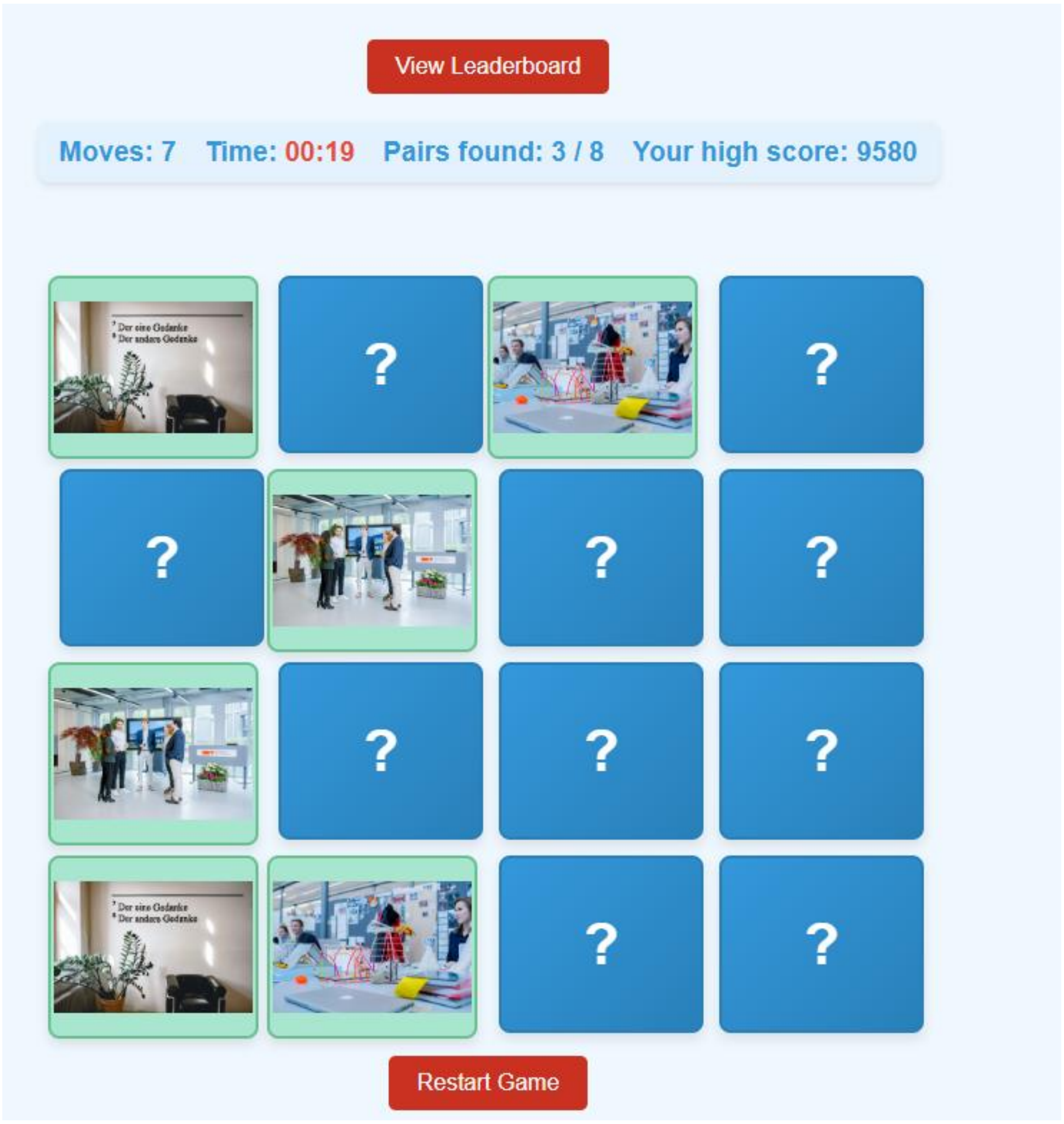
Start game

Once the whole login process is done the user is shown the home / index screen:


Welcome to the memory game of TriMy!

You will be rewarded points based on the time needed and the amount of moves needed to find all eight pairs

In some cases, the user will see a small loading indicator before he/she is able to start the game. But as soon as all the pictures are fully loaded the game can be played:



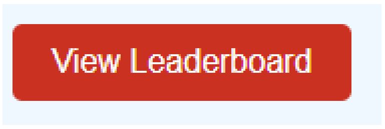
In the game info screen the player can see some stats for his current game as well as his personal high score.

Restart game

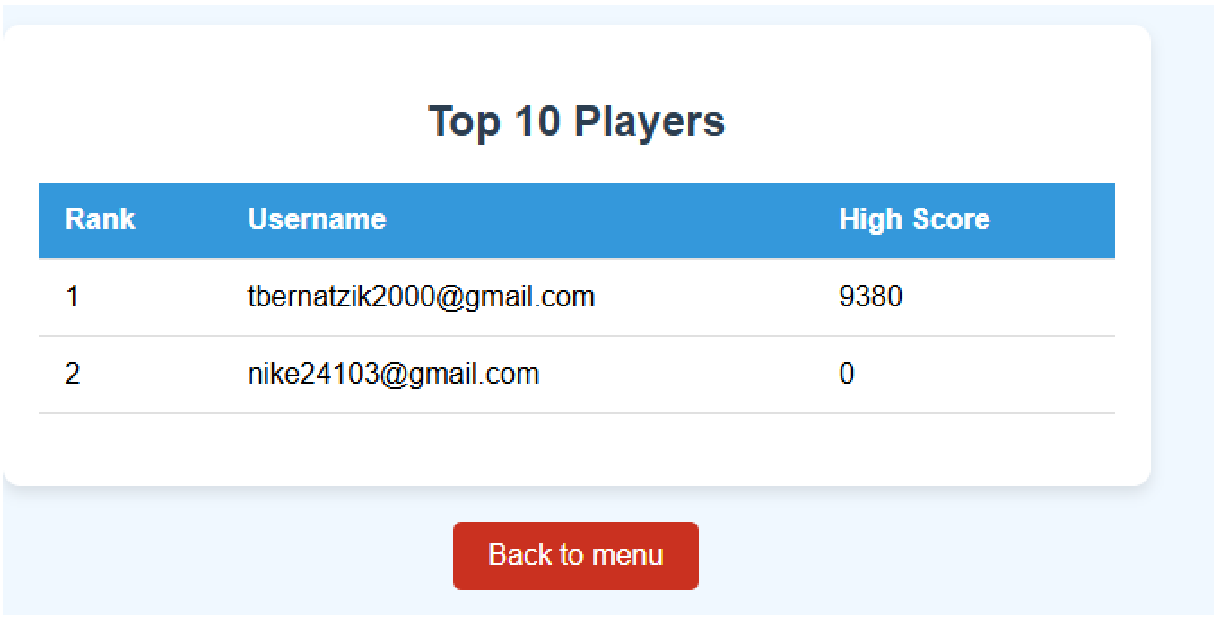
After a game is finished the user needs to click on the "Restart Game" button to start his next attempt. But if he/she is not happy with his/her current attempt, this button can be used to just start a new attempt right away without finishing the current one.

Check leaderboard

To compare his/her personal high score to others the user can check the leaderboard to see if his highest score is in the top 10. he/she can do that by clicking here:



There the rank, username and the high sore are shown:



Top 10 Players

Rank	Username	High Score
1	tbernatzik2000@gmail.com	9380
2	nike24103@gmail.com	0

Requirements

In this section we will list everything that is needed to make sure our app and all its additional files can be run accordingly. In the Hosting part we will explain how the hosting on PythonAnywhere works.

Libraries and packages

In our app.py file we use the following import statements:

```
import random
import os
import uuid
from flask import Flask, render_template, request, session, jsonify,
redirect, url_for
from flask_login import LoginManager, login_user, login_required,
logout_user, UserMixin, current_user
from werkzeug.security import generate_password_hash, check_password_hash
from flask_sqlalchemy import SQLAlchemy
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail
from dotenv import load_dotenv
```

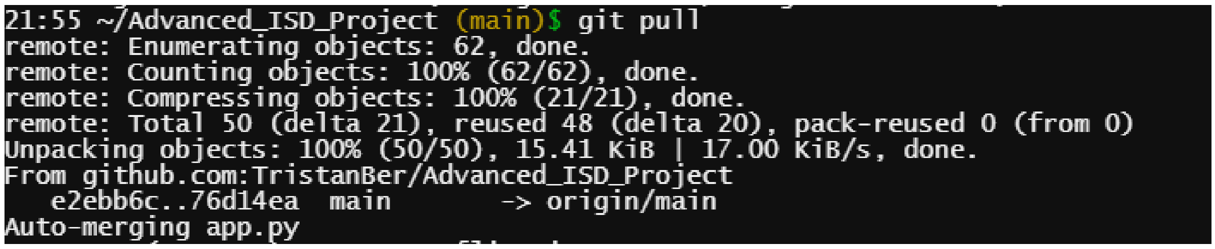
To let us import them into our virtual environment in PythonAnywhere we created a requirements.txt file which we added to our git:

```
# requirements.txt
# List of Python packages required by the application.
# Use 'pip install -r requirements.txt' to install them.

Flask
Flask-Login
Flask-SQLAlchemy
sendgrid
python-dotenv
```

Hosting

While we have developed and tested our code locally on our own machines and committed our changes to GitHub, we still needed a way to publish our app on a website. Initially we tried to do so with GitHub pages, but we determined that PythonAnywhere was more suitable considering our planned functionalities. We have connected our GitHub repository with PythonAnywhere with an SSH key. This allows us to simply pull our changes from GitHub to PythonAnywhere directly from a bash-console:



```
21:55 ~/Advanced_ISD_Project (main)$ git pull
remote: Enumerating objects: 62, done.
remote: Counting objects: 100% (62/62), done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 50 (delta 21), reused 48 (delta 20), pack-reused 0 (from 0)
Unpacking objects: 100% (50/50), 15.41 KiB | 17.00 KiB/s, done.
From github.com:TristanBer/Advanced_ISD_Project
   e2ebb6c..76d14ea  main       -> origin/main
Auto-merging app.py
```

As we developed the login and verification part, we needed some packages which are not available in PythonAnywhere by default. To fix this we had to create a virtual environment in PythonAnywhere and install the packages (which are described in above).

To make sure the website stays up and running it is needed to activate it every three months, otherwise the server will be taken down – it can then be activated again without any further ado. If we want to let it run without this "activation" we would need to pay a subscription fee.

We use the EU version of PythonAnwhere (https://trimy.eu.pythonanywhere.com/) to make sure the server which hosts our website is running in Europe instead of the US.

Database setup

We use an SQL database (SQLite version) to store the information about the user: username (email), the password-hash and highest score. The future development is to store the information of the highest score depending on the mode the user played in. We execute this by using SQLAlchey, a package that integrates with the Flask. It makes the process easier and demanded less time to program the process of storing the data. The database is obviously not visible to the user, but some of it is. We established the leaderboard, which excerpts in real time the data of highest 10 scores across all the players (by real time we mean per moment, when the user press "View Leaderboard" button).

To make sure we do not overwrite the database file stored in PythonAnywhere every time we pull new changes, we setup a .gitignore file which contains the instances/user.db file.

SendGrid

This is a third-party application that we use to send automatically emails with confirmation codes. The best-case scenario is when we have own domain. Then more options for setup are available, as well as such emails can look more professionally: i.e. no-reply@trimy.game.

Yet, we used a free-to-use approach: we authorised ourselves with private email (trimygame@gmail.com). This is a working solution, but not for a long-term. For it to work we had to obtain an API KEY and integrate it into our main Python script. Then approve the email from which the verification is sent.

Known issues

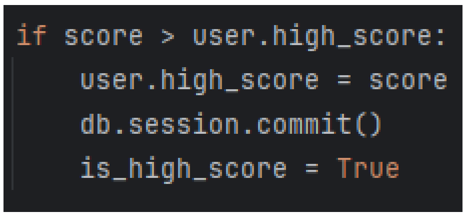
Here we will list all issues that we currently know about. If new bugs are found over time, we can add them here and check how we are able to fix them.

Performance

Especially in the beginning of the project we faced some performance issues when playing the cards as the pictures were loaded one by one when flipping the cards. To counter this, we implemented a preloading function that makes sure the game can only be started once all pictures are fully loaded. In some cases, probably when the browser tries to load the pictures from the cache the first card still takes longer to load than the rest of the cards. But latest with the second round this is no issue anymore and the game runs perfectly smoothly

User high score NULL

In the save_score route / function we are checking if the score which the user achieved in this session is higher than what he/she achieved before in this if-statement:



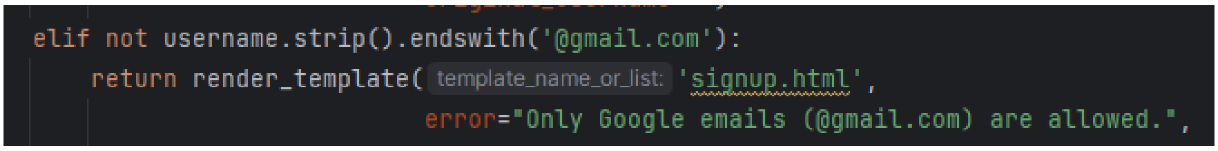
```
if score > user.high_score:
    user.high_score = score
    db.session.commit()
    is_high_score = True
```

If the stored value in the database is “NULL” this check will run into an error – no matter what his new score is. We decided not to check how we can fix this as the only case this can happen is when someone deletes the entry directly in the database. The default value for the high_score field in the database is 0. We only found this bug because we tried to reset a user's score in the database.

Verification only for gmail.com mail addresses

As the original idea was to make our website only available for users with an active uni.li email address we implemented a logic which makes sure the entered email address ends with @uni.li. Unfortunately, the verification mails which were sent by SendGrid were blocked by the university server.

We also encountered this issue that no verification code is received when we tried to send it from the uni.li email.

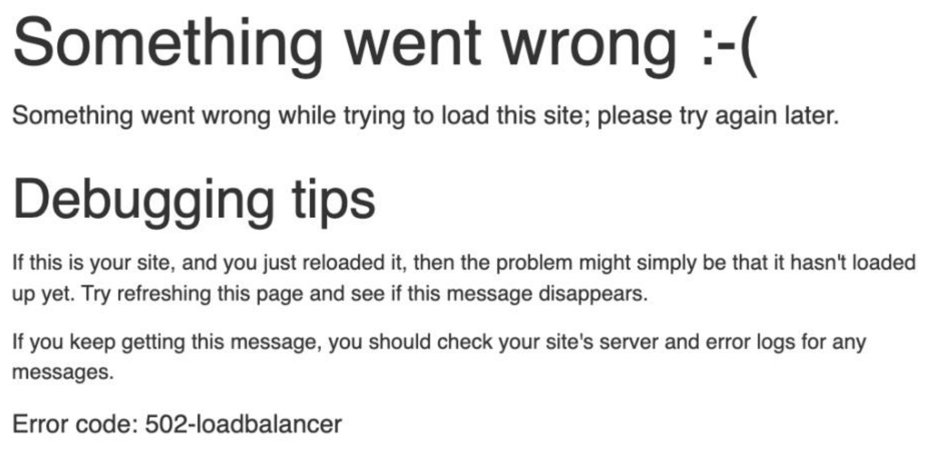


```
elif not username.strip().endswith('@gmail.com'):
    return render_template( template_name_or_list: 'signup.html',
                           error="Only Google emails (@gmail.com) are allowed.",
```

We could also allow all mail addresses to be verified but to show our idea of only providing it to uni.li accounts we decided to restrict it to gmail.com users.

Error after multiple games

In some cases, test users experienced an issue after making multiple games in a row:



Something went wrong :-(

Something went wrong while trying to load this site; please try again later.

Debugging tips

If this is your site, and you just reloaded it, then the problem might simply be that it hasn't loaded up yet. Try refreshing this page and see if this message disappears.

If you keep getting this message, you should check your site's server and error logs for any messages.

Error code: 502-loadbalancer

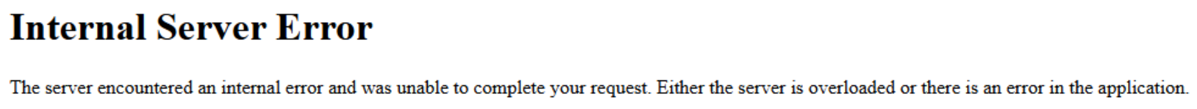
This occurred while using Safari and Google Chrome. We tried to fix this by implementing a function which checks how many sessions are currently being stored and makes sure only a maximum of 3 are saved. Since then, this issue has not reoccurred anymore but since we only have limited resources available in PythonAnywhere because we are using the free version it could be that this issue comes up again in case more users are playing simultaneously.

Password complexity

We are not checking if the password provided by the user during the signup is meeting any complexity standards. Therefore, users can enter any password they want. As no sensitive information are stored on our website, we don't think this is very critical.

Reset password or account

Currently it is not possible to reset a user's password in case he/she forgets it. It is also not possible to re-register with an already used email. Even though the verification code is sent when the user tries to enter it, he/she receives an internal server error:

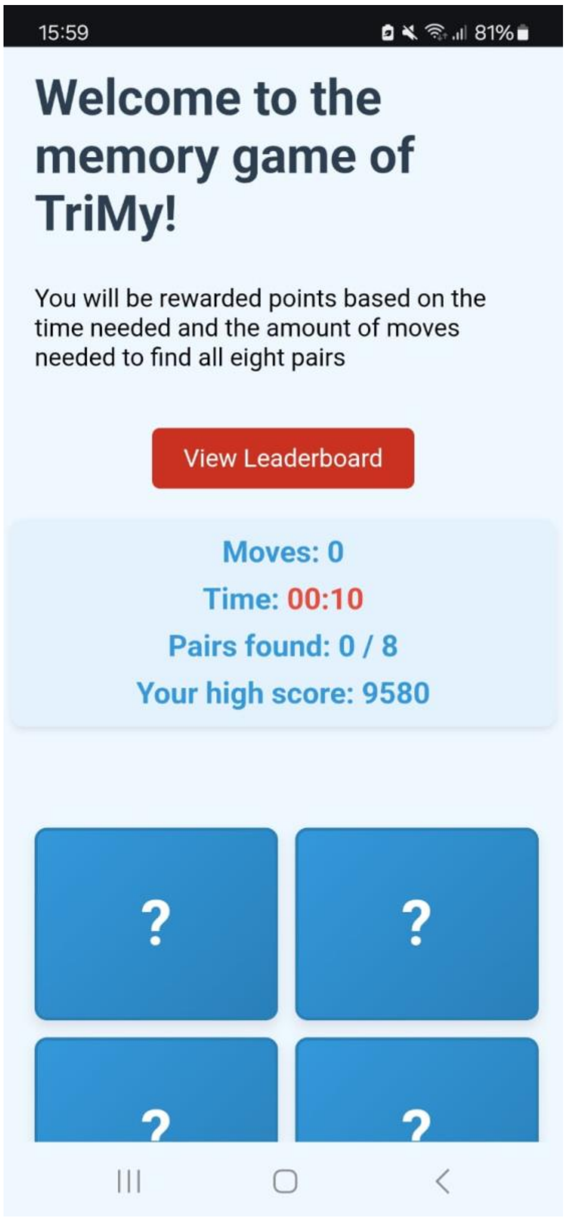


Internal Server Error

The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.

Mobile usage

We did not focus on using our app on mobile devices, which normally have much smaller screens than laptops. Even though we made sure our style is responsive regarding the screen size it is not optimized for mobile devices:


15:59 81%

Welcome to the memory game of TriMy!

You will be rewarded points based on the time needed and the amount of moves needed to find all eight pairs

Moves: 0

Time: 00:10

Pairs found: 0 / 8

Your high score: 9580

Multi-mode

By multi-mode we mean a game set-up, where the player selects with how many pairs of cards, he/she wants to play. So far, we have ready modes with 8,10 and 12 pairs: this includes adjusting the code for player to be able to choose, as well as game logic, changing the looks of the game tables, creating new interface for the player to choose the number of pairs.

We, however, encountered following issues that stopped us from releasing the version so far: 1) issue with scoring calculation; 2) leaderboard. As for the second point, the issue with the leader board is that we have to split now the tables to depict different scores for the same user from different modes. We expect this issue to be resolved by the presentation date.

Version Control

To make sure we can work and develop our project simultaneously we decided to use GitHub as our versioning tool. We always developed and tested our code locally and then committed and pushed our code to GitHub. From there we took it over PythonAnywhere.

The entire history of our changes can be found in our GitHub repository.