KCL Buddy Scheme

Dr Jeroen Keppens

Department of Informatics, King's College London

Team Null Pointer Exception:

A K M Naharul Hayat

Noyan Raquib

Oto Mraz

Gavir Virk

Dusan Pilka

Ian Luong

Andrea Covasa

Introduction

The business objective is to run a university buddy scheme where senior students are paired up with junior students in the same session/scheme in the form of a buddy/mentor, in order to provide support to them throughout their time at university. The aim of the project is to automate administration of aspects of the system such as sign up, verification of participants, allocations of buddy/mentor to students and monitoring/evaluating the system. The purpose of this project was to make a flexible and extendable application as the buddy scheme is new and will develop significantly in the coming years.

Outcome

In order to meet the client's business objective, we have developed a web application to administer/manage the buddy system. The important high-level features of the solution we have developed is as follows:

Student/User Area

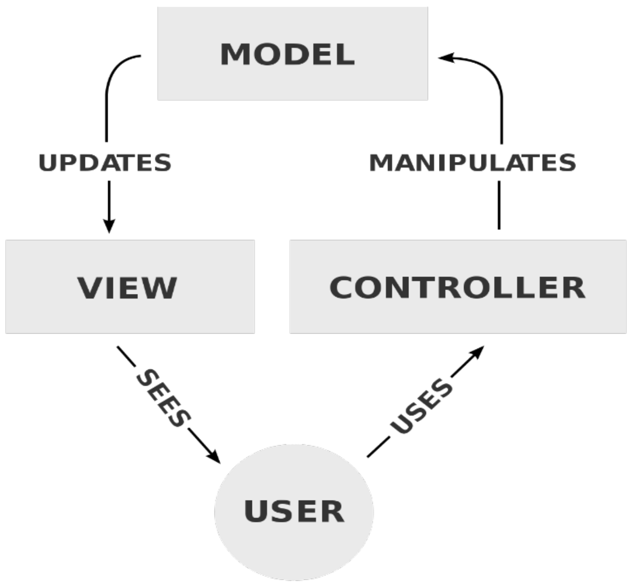
- Participants are able to create an account/register either via the website itself or via a specific registration link for a particular session/scheme. They can also reset password for their account.
- After creating an account, they are able to verify their identity via their K-Number through a verification email.
  - Unless a user is verified, they cannot be allocated a match.
- Once the user is verified, they can sign up formally to sessions/schemes that they are a part of by completing a short profile.
- If the student created an account or logged-in via a custom registration link for a session/scheme, they do not have to search for their session/scheme as it will already be listed in the dashboard making it easier to complete the profile.
- Students can update/change their profile for a session/scheme until they have been allocated a match, but not after.
- The student can view the sessions/schemes they have signed up in along with a status for each of them, displaying whether they have been allocated a match yet or not.
- Once a match for the user has been allocated, they are sent an email informing them to log in their account to view their buddy allocations. Once they log back in, they can view name/contact/email/profile description of their allocated buddies.

Admin Area

- Super-admins have all the privilege. They can create/manage all sessions/schemes.
  - Super admins can also create & manage regular admins and allocate particular sessions to them. The regular admins are **only** able to manage sessions allocated to them by super admin. They also cannot manage admins like super/admin can.
  - We have implemented and tested middleware to guard against unprivileged access.
- Admins are able to create and manage a set of interest choices and allocate choices to each sessions/scheme.
  - Students signing up to the sessions/schemes will have the set of interest/hobby choices for that particular session/scheme (set by the admin) to choose from. Admins can also choose to not have pre-listed interest choice options for a particular session.
- Admins can view register links for each session/scheme in order to distribute them.
- Admins can send feedback email containing survey link to students. They are able to use an external survey website integrated in the application, in order to view/evaluate survey results.
- Admins can view allocations for the sessions and are able to allocate and deallocate matches manually.
  - When allocating manually, they are only able to allocate to seniors who are eligible to be allocated to.
  - The information also displays number of common interests and adds data filtering options in order to make the manual allocation process easier.
- They can also run the matchmaking algorithm which matches students by taking into account their interests in common and other constraints such as same gender allocation preference or special needs priority allocation. They can also reset matches, provided they are not finalized
- In addition, they can finalize matches and send out emails informing students of their allocation.
  - However, once a match has been finalized and emails sent out, they cannot deallocate that match.
- After allocations are done, admins can also lock a session after deadlines in order to disable further students signing up to a particular session.
  - They can also then send out an apology email to all students who couldn't be allocated by the click of a single button.

Design

We have chosen to adopt the Model-view-controller (MVC) architecture. This is among the most popular web development frameworks in use today and offers excellent separation between the front and back-end. The diagram below provides a brief overview of the components of this architecture:



As seen in the diagram above, the front-end elements (views) are completely separated from any program logic and database queries. This improves design by ensuring better encapsulation and modularization of components. This separation also allows, for example, one person to contribute to the front-end component and another person to work on back-end PHP/MySQL elements in parallel with minimal conflict.

Framework

We used the Laravel framework to enforce MVC architecture of the project.

- Laravel separates the views, controllers and models into separate directories to allow for minimum coupling between components and greater maintainability.
- Laravel also provides its own implementation of security, which we can simply extend to tailor according to our needs.
- Laravel provides its built in eloquent ORM, which allows seamless interaction with the MySQL database.
- We have used Laravel's "blade" template engine for the views. Using blade allows to effectively ensures that we do not mix any html and php code in the views. As a result, it allows us to ensure separation between front and back end components. Finally, it makes the code in the view components more readable.

Use of models and database queries

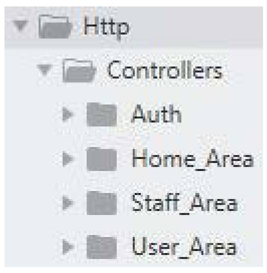
We have abstracted the interaction with the database via models. Each database table corresponds to a model in our application, and we use that model to interact or perform query and CRUD operations in the database. Models are located at – **App**

- We have used Laravel's own built in ORM known as 'eloquent' and query builder to interact with the database models. This makes our query code more readable, clear and concise.
- Query functions are encapsulated in their respective models. This allows us to re-use query functions when we need to and eliminates code duplication.
- A model which abstracts a database table, only contains query function which relates primarily to that table in the database. This makes our application more cohesive.

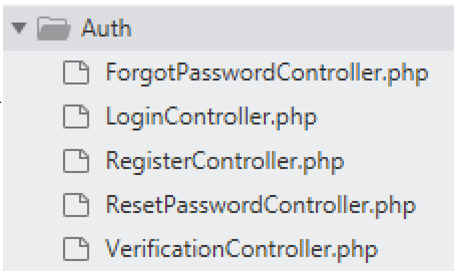
Controller design and directory structure:

The job of the controllers of our application is to instantiate the models and retrieve data through it and pass it into the views. We do not execute query functions in the controllers. Therefore, the controllers are perfectly cohesive. This further enhances maintainability of our application.

- We have divided the directory structure of the controllers of our application as follows



For Example

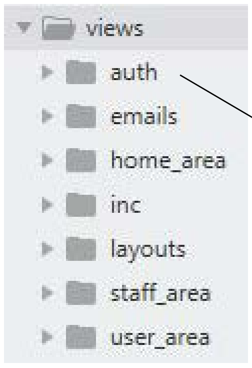

Auth
ForgotPasswordController.php
LoginController.php
RegisterController.php
ResetPasswordController.php
VerificationController.php

In each parent directory of controllers, we have controllers only concerning to that particular area only

a single controller is only responsible for its respective area/view. this makes it easy to understand and make changes to different areas of the application. This also reduces coupling between the different components of the application

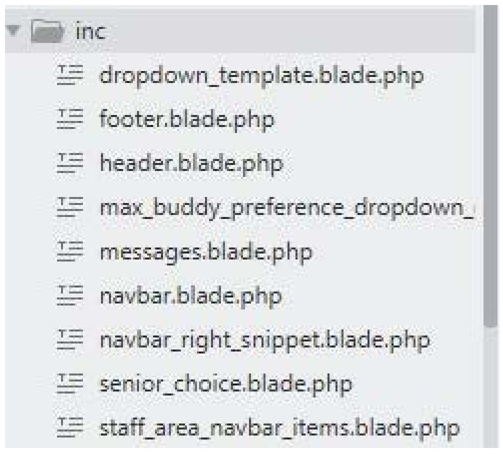
- All variables and methods have been made private where possible and we have tried to protect access as much as possible using getters and setters. Methods in our classes are only public if there is a reason to call them from other classes and they are not implementation specific.
- Controllers are located at – **App\Http\Controllers.**

Preventing code duplication and maximizing cohesion in Views:



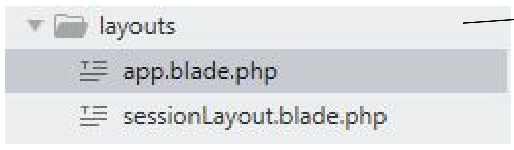
We have split view components of our application into separate folders, each containing views representing its respective area. The directories are further split down the line in order to make the structure more cohesive.

For example, views concerning authentication (registering, login etc) are confined in the Auth folder, and so on.


inc
dropdown_template.blade.php
footer.blade.php
header.blade.php
max_buddy_preference_dropdown_
messages.blade.php
navbar.blade.php
navbar_right_snippet.blade.php
senior_choice.blade.php
staff_area_navbar_items.blade.php

Inside this folder, we have separated re-useable code of our views into separate files. So, when we have to use it, we can just include that file easily, instead of re-writing that code. This eliminates code duplication in the view components of our application

For example, the 'navbar.blade.php' file contains code only concerning the navigation bar of our application. if we have to change the navigation bar, we can just make the changes here and it would affect the entire application. This makes the code more maintainable and extendable and if changes are needed, we only need to change confined in a single file



Furthermore, we have abstracted and put together single cohesive and consistent layouts for views across our application. We include the layouts in each views of the application and this reduces code duplication and maintains consistencv

The view files for our application can be found at – **resources\views**

Minimizing Code Duplication through inheritance of controllers:
In order to prevent code duplication: Inheritance and abstract classes are used where appropriate. An example where this is applied is provided below:

- For Create, Read, Update and Delete (CRUD) operations in the Admin Area of the application, we have created an abstract class called CRUDController.
    - This abstract class contains general implementation of methods concerning CRUD operations. Classes inheriting from this class, is required to implement the abstract methods defined. One such method is the getModel method, which allows the sub-class to pass in the model on which CRUD operations are to be performed
    - The Sessions, Interests and ChangeAdmin controllers all inherit from this class, and mostly uses the general implementation logic in the superclass
    - Where some logic changes are needed, the respective methods are overridden, but most of the core logic remains in the CRUDController superclass
    - This minimizes code duplication in our application and makes our application more extendable. In the future, if we have to perform CRUD operations concerning some other model/components, we can just inherit from the superclass, and don't have to implement the core CRUD functionality again.

A similar inheritance structure is used to prevent code duplication for Profile CRUD operations and for the email functionality in the application.

CSS styling unification

We have also separated and unified CSS styling across the entire application in a single CSS file (**located in public/styles.css**) and have referenced all UI components in the application via their respective id.

- Therefore, if styling of a specific component has to be changed, for example a button, they can all be done in this file, and it would affect buttons across the entire application for a consistent look and feel.
- We do not have any in-line CSS code in our views, as this creates inconsistency in styling and makes styling changes a hassle.

Validation Rules Modularization

To better modularize custom validation rules for forms. We have created form verification rule classes which implements the 'Rule' class in Laravel framework.

- This enables us to re-use custom validation rules in different forms and eliminates code duplication. These rule classes are located at – **App\Rules.**

Separation of JavaScript

We have separated our JavaScript files into a separate folder and have only linked them to the view. Therefore, the view files do not contain any JavaScript code/logic. They are completely separated into separate folders and files. The JavaScript files are located in – **public\js**

Security

Laravel also provides us with a comprehensive security layer allowing us to easily define user authentication. It enables protection against security vulnerabilities such as SQL injections, cross site request forgery (CSRF) etc.

- The security middlewares are implemented in separate classes to better modularize the code. These are located in – **App\Middleware.**
- We have also hashed passwords in our database and have tested middleware redirection. These tests are located at - **tests\functional\Security**

Front-end framework

We use Bootstrap to deliver our front-end components. Bootstrap has been shown to be a highly efficient and scalable framework in industry.

- Bootstrap's responsive CSS adjusts to phones, tablets, and desktops. In addition, it is compatible with all modern browsers (Chrome, Firefox, Internet Explorer, Safari, and Opera)

Routes

We have used Laravel routes to link URLs with specific controllers. The routes for our application can be found at – **routes\web.php.**

Efficiency
Since the buddy scheme is expected to grow significantly in the future. We could have thousands of students signing up and our backend components should be able to scale well according to that.

- A crucial design decision catering towards the efficiency needs of the application was to do most of the expensive data processing for the matchmaking algorithm primarily via SQL queries. Since SQL is specialized for data processing/retrieval, we take advantage of it to execute most of the expensive data processing for the matchmaking algorithm.
- Therefore, the backend components of our application will scale well. Even if we had thousands of students signing up for the buddy scheme, the algorithm will be able to allocate matches in a reasonable amount of time.

Robustness
- Our application employs very thorough data validation on all user inputs. Our automated testing with well over 400 assertions shows that data validation is indeed enforced.
- If somehow an error does occur due to an update to the application by a future developer, appropriate error message is displayed by Laravel itself.
- We have imported libraries in our code instead of relying on Content Delivery Network (CDN)

Testing

Testing Approach
We employed an automated testing approach. This enabled us to test our application quickly and easily each time we added a new feature to it. A full set of test results for our application is shown in the appendix of the report.

Framework/tools
We have used a testing framework known as **'Codeception'** which integrates very well with the Laravel framework. It provides 3 separate test suites split into their respective directories and we have taken advantage of each of them in order to test our application thoroughly:

- **Unit Tests:** We used these tests to test individual isolated components/functions/methods of our application. These tests can be found at - **/tests/Unit**
- **Functional Tests:** We used these tests to test various component interaction of the application and Create, Read, Update and Delete (CRUD) operations on the database. These tests can be found at - **/tests/functional**
- **Acceptance Tests:** These tests run in real time under real circumstances by Emulating a real browser known as 'PHP Browser'. The server needs to be running in order to run these tests. These tests can be found at - **/tests/acceptance**

Testing Strategy
We have based our testing strategy in terms of the use cases of the application and we prioritized edge cases in scenarios when writing tests. For each of the use cases of the application, we have written tests to cover the following scenarios:

- **Functional & Unit tests**
  - **Positive case -** The expected scenario from the user
  - **Negative Case -** Scenarios which should not be accepted in the system and should display an error of some sort for the user. i.e.: scenarios designed to test violations.
  - **Boundary Case -** We test the boundary scenario of the use case, i.e.: right on the edge of what should be accepted.
- **Acceptance Tests**
  - As these tests emulate a real web browser, we tested the view components of the application, and interaction with various components in the page by the user and appropriate redirections by the user.
  - We have also tested security middlewares using acceptance tests and their resulting redirection and error message. For example, we have tested cases where a regular admin tries to go to the super admin area by manually entering the link.

Testing - Important Note

We have used a separate database instance for testing during development, and a separate one for the deployed version. We have kept the test database constant throughout development and have taken advantage of the automated testing to verify correctness of behavior after making changes.

- Therefore, please note that, development version which is linked to the test database has some tests which rely on specific data records in the test database (for example, super admin login, student records etc).
- Since our test database is meant to be constant, making changes to some test data during marking may misleadingly show test failure, **however, having a constant database with content specifically built entirely for testing purposes was an intentional development decision, which has helped our team verify correctness of behavior throughout the development efficiently.**
- Due to the nature of our application which is mostly database CRUD operation and front based, we have mainly relied on functional and acceptance tests. By using functional tests, we could test efficiently whether CRUD operations were successful or not. Furthermore, functional tests in Codeception resets database changes resulting from tests automatically after tests are completed.

Running the tests

To run the development version of the application, you must have composer installed. After this, ensure you are in the project directory and run:

composer install

After this, simply run the following command, and keep the terminal running:

php artisan serve

This will give an URL, usually **http://localhost:8000**, which you can copy and paste into the browser to start the application.

To run the tests, again ensure you have composer installed and php artisan serve running, then execute:

On Windows - **vendor\bin\codecept run**
On Linux - **vendor/bin/codecept run**

This will run all the tests we have available, including all: unit, functional and acceptance tests. Bear in mind this can be a bit time consuming and can last up to 5 minutes.

Team organization

At the start of the project, we identified core expertise of individual members of the team and have allocated tasks/roles based on it. Throughout the project, we have combined and utilized knowledge/expertise of all individuals in order to build the solution for the client. The following table briefly demonstrates core contribution areas of different members of the team.

Team member	Role	Contributions
A K M Naharul Hayat	Full Stack + Client Liaison	Integration of front-end with backend, Authentication, CRUD operations, Database queries, UI implementation, Form Validations, K number verification, Code refactoring, Table pagination, Table data filtering, Report, Testing.
Oto Mraz	Full Stack + Client Liaison	Email functionality, code refactoring, CRUD operations, database queries, Integrate external survey website, Report, Testing.
Noyan Raquib	Front-end + Deployment + Client Liaison	Front-end component design and Implementation, Form data preselection, Cookie implementation, Stylings, JavaScript, Report, Deployment, Testing.
Dusan Pilka	Back-end	Design and implement matchmaking algorithm, Design database schema.
Gavir Virk	Back-end	Database query, CRUD operations, Refactor code, Testing.
Ian Luong	Front-end + Testing	Integration tests, Unit tests, Risk analysis, screencast presentation.
Andrea Covasa	N/A	N/A

Other Pertinent Information

Login Credentials:

Super-admin logins:

- **email:** rob@swire.com **password:** 123456
- **email:** major.test1234@gmail.com **password:** Major1234
- To test student users during marking, we can simply register as new student and super admin can be used to create regular admins. Admins can also generate registration links for specific sessions for students.

Deployed Version Link - https://kclbuddyscheme.000webhostapp.com/

SurveyMonkey Login: Username: MajorProject123 Password: Major123

Link to shared Google Drive folder for documentation of the project:

https://drive.google.com/drive/folders/1EcHFmjZYMbLWnlsy053NjtOTtzHwusIw?usp=sharing

Running the application:

To run the development version of the application, you must have composer installed. After this, ensure you are in the project directory and run: **composer install**

After this, simply run the following command, and keep the terminal running: **php artisan serve**

This will give an URL, usually **http://localhost:8000**, which you can copy and paste into the browser to start the application.

Risks that materialized

We had one major risk materialize in our project. We had initially decided that such a risk would have a low probability, but serious consequences. This risk was that one of our team members made virtually no contribution to the end product, having 0 commits on GitHub. This member also did not make any contribution to any report produced by the team. This essentially meant we had an effective team of 6 rather than 7.

During the risk mitigation stage, we had decided that we would reduce the effects of this (along with a number of other) risks by aiming to finish all deliverables within 2-3 weeks and if this specific risk did materialize, we would relocate existing members within the project, as we have multiple people who were proficient in both the front and the back end.

When this risk materialized, we followed our risk mitigation strategy. We gave some extra responsibilities to existing members working on the front-end and re-assigned some members working on other parts of the project to front-end design. However, this risk did mean that we had effectively finished all major parts of our project with 1 week remaining rather than 2-3 weeks. Nevertheless, since our team had written tests for our key deliverables at the same time as writing code, we managed to finish out project in good time. Also, the code quality of most team members was better than expected (in part due to most members strictly following Laravel and MVC conventions), so there was limited need to refactor the code towards the end.

Difficulties/Challenges during the project

We did not have any significant difficulties during the project. However, we did find that some of the APIs and frameworks more difficult to use than otherwise expected. One example of this would be that since Laravel used Bootstrap 4 by default on the latest version, several features from Bootstrap 3 no longer worked. For example, glyphicons were supported in Bootstrap 3, but not in Bootstrap 4. This meant we had to use a mixture of Unicode elements and small images, which were far less convenient than glyphicons.

In addition to this, the select picker API was unable to detect the version of bootstrap we were using, which meant we had to manually import the JavaScript and CSS for this API and set the version of bootstrap we used within those source files. We have found this to be quite time consuming, giving the length of those particular API source files.

APIs and Frameworks used in the project

- Bootstrap 4
- Laravel
- Bootstrap select picker - https://developer.snapappointments.com/bootstrap-select/
- jQuery 3.3.1
- Codeception

Relevant source files for select picker used from:

https://cdnjs.cloudflare.com/ajax/libs/bootstrap-select/1.13.8/css/bootstrap-select.css

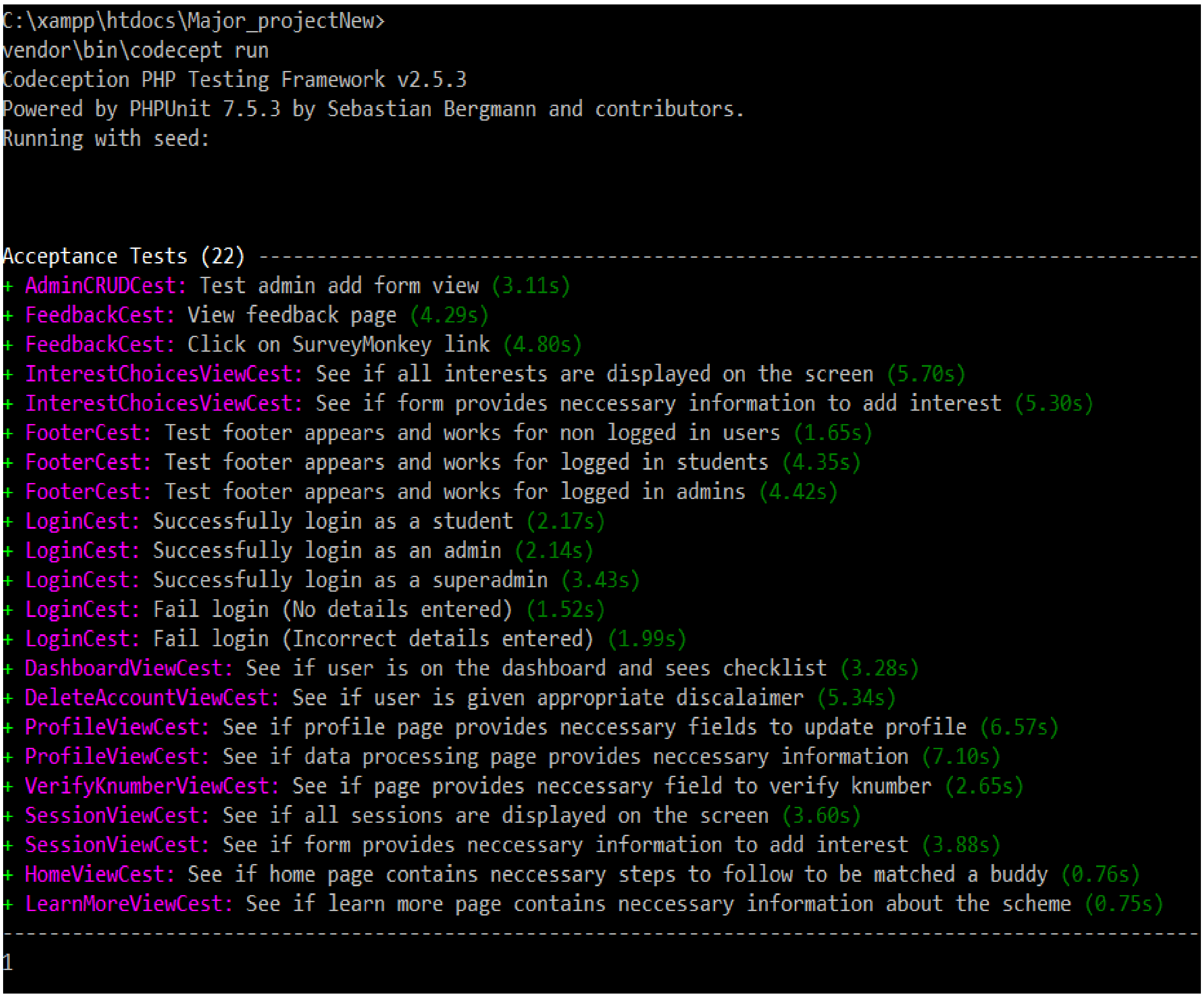
https://cdnjs.cloudflare.com/ajax/libs/bootstrap-select/1.13.8/js/bootstrap-select.js

Appendix

Test Results

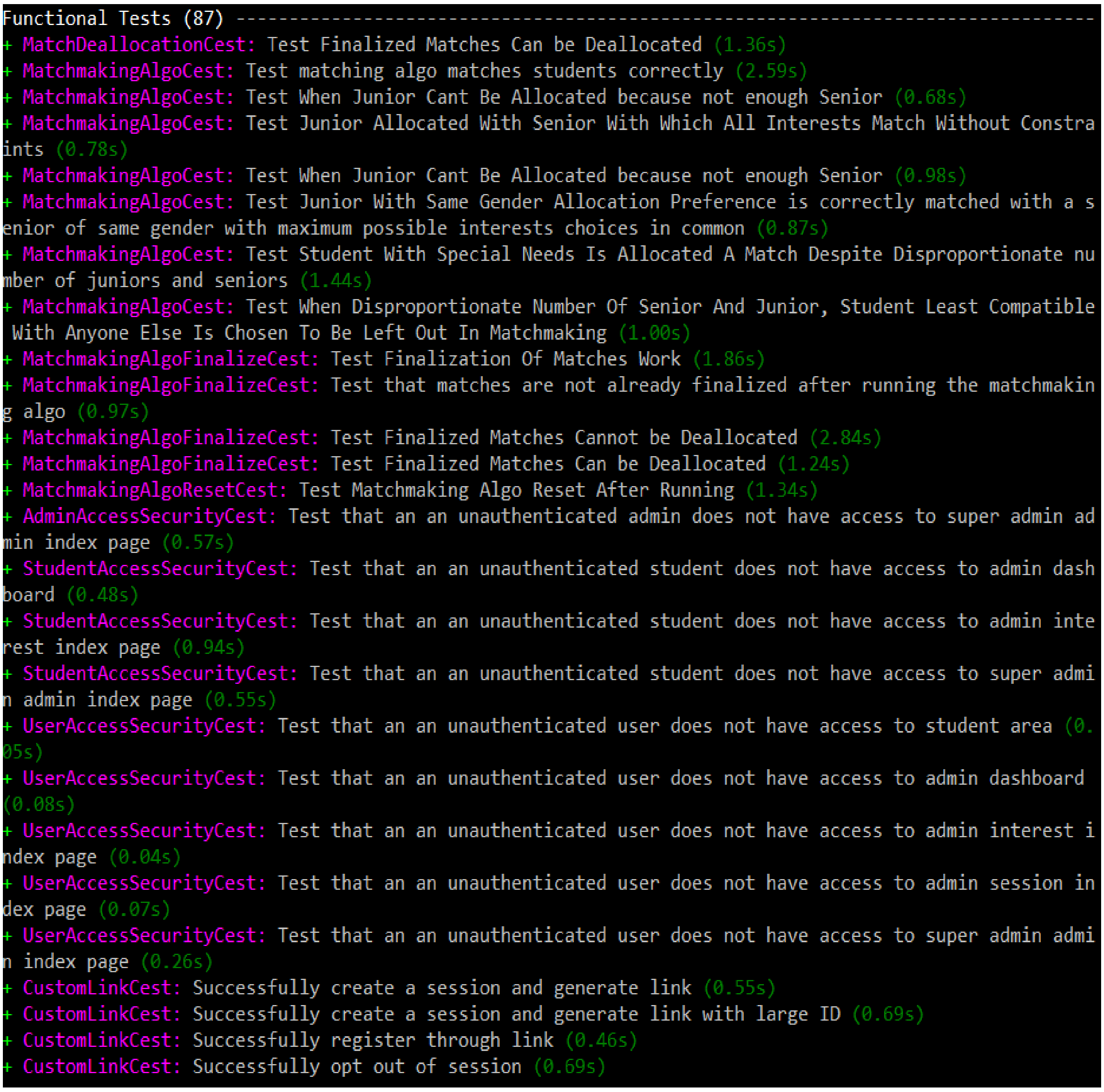
The images below show the command line output, when running the automated tests. They are grouped into three sections: acceptance, functional and unit. The green '+' symbol in front of the test means that the test has successfully passed (a failure would be indicated with a red '×' symbol). Then we have the test class name (in pink) and the name of the test. Lastly, we can see the time taken to execute the test. At the end of the tests we have a summary indicating the total time taken (3.03 minutes), the total number of tests executed, and number of assertions made. The word 'OK' highlighted in green shows that all tests have passed.

Acceptance tests



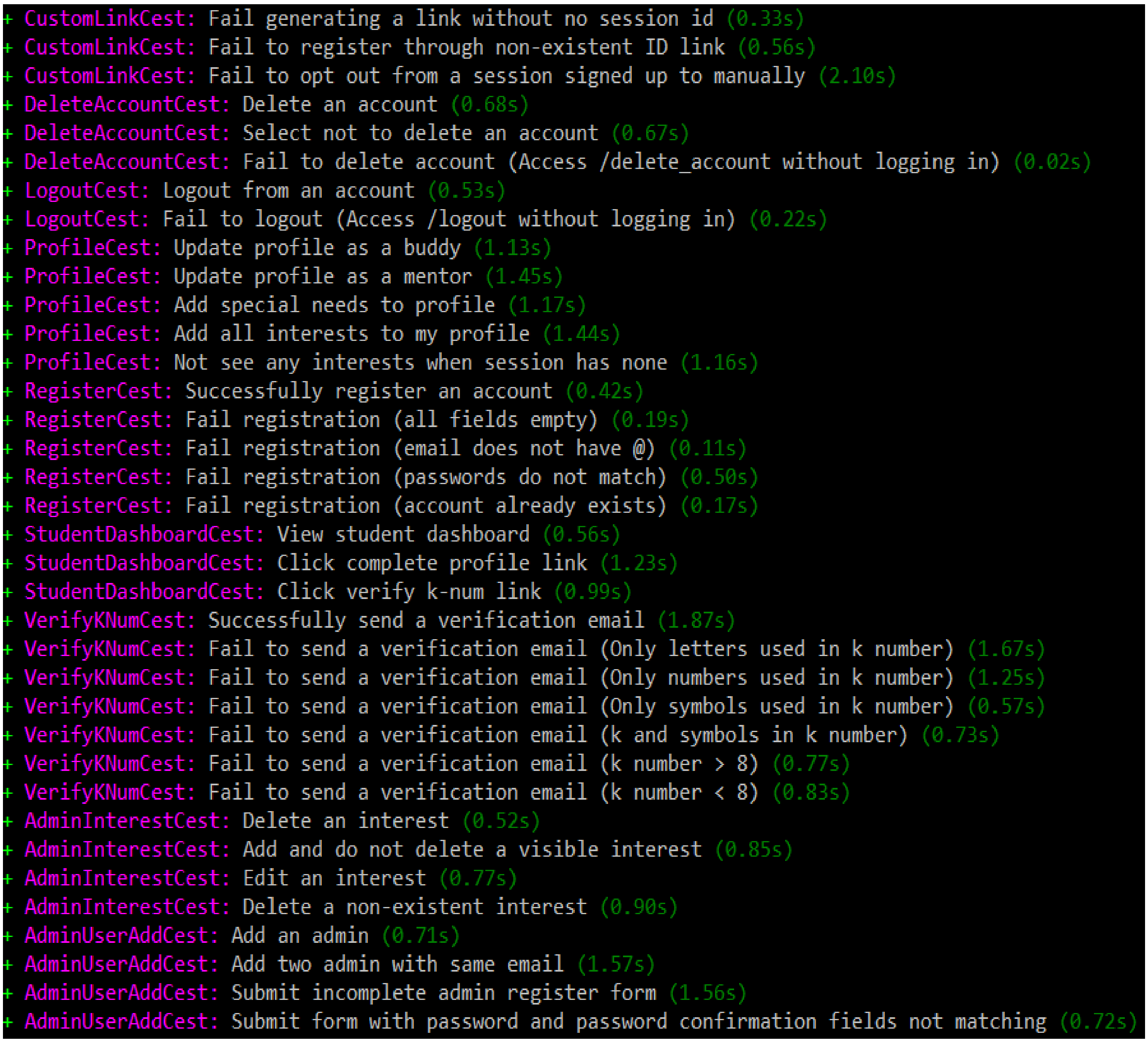
```
C:\xampp\htdocs\Major_projectNew>
vendor\bin\codecept run
Codeception PHP Testing Framework v2.5.3
Powered by PHPUnit 7.5.3 by Sebastian Bergmann and contributors.
Running with seed:



Acceptance Tests (22) ---------------------------------------------------------------------------
+ AdminCRUDCest: Test admin add form view (3.11s)
+ FeedbackCest: View feedback page (4.29s)
+ FeedbackCest: Click on SurveyMonkey link (4.80s)
+ InterestChoicesViewCest: See if all interests are displayed on the screen (5.70s)
+ InterestChoicesViewCest: See if form provides neccessary information to add interest (5.30s)
+ FooterCest: Test footer appears and works for non logged in users (1.65s)
+ FooterCest: Test footer appears and works for logged in students (4.35s)
+ FooterCest: Test footer appears and works for logged in admins (4.42s)
+ LoginCest: Successfully login as a student (2.17s)
+ LoginCest: Successfully login as an admin (2.14s)
+ LoginCest: Successfully login as a superadmin (3.43s)
+ LoginCest: Fail login (No details entered) (1.52s)
+ LoginCest: Fail login (Incorrect details entered) (1.99s)
+ DashboardViewCest: See if user is on the dashboard and sees checklist (3.28s)
+ DeleteAccountViewCest: See if user is given appropriate discalaimer (5.34s)
+ ProfileViewCest: See if profile page provides neccessary fields to update profile (6.57s)
+ ProfileViewCest: See if data processing page provides neccessary information (7.10s)
+ VerifyKnumberViewCest: See if page provides neccessary field to verify knumber (2.65s)
+ SessionViewCest: See if all sessions are displayed on the screen (3.60s)
+ SessionViewCest: See if form provides neccessary information to add interest (3.88s)
+ HomeViewCest: See if home page contains neccessary steps to follow to be matched a buddy (0.76s)
+ LearnMoreViewCest: See if learn more page contains neccessary information about the scheme (0.75s)
------------------------------------------------------------------------------------------------
1
```

Functional Tests



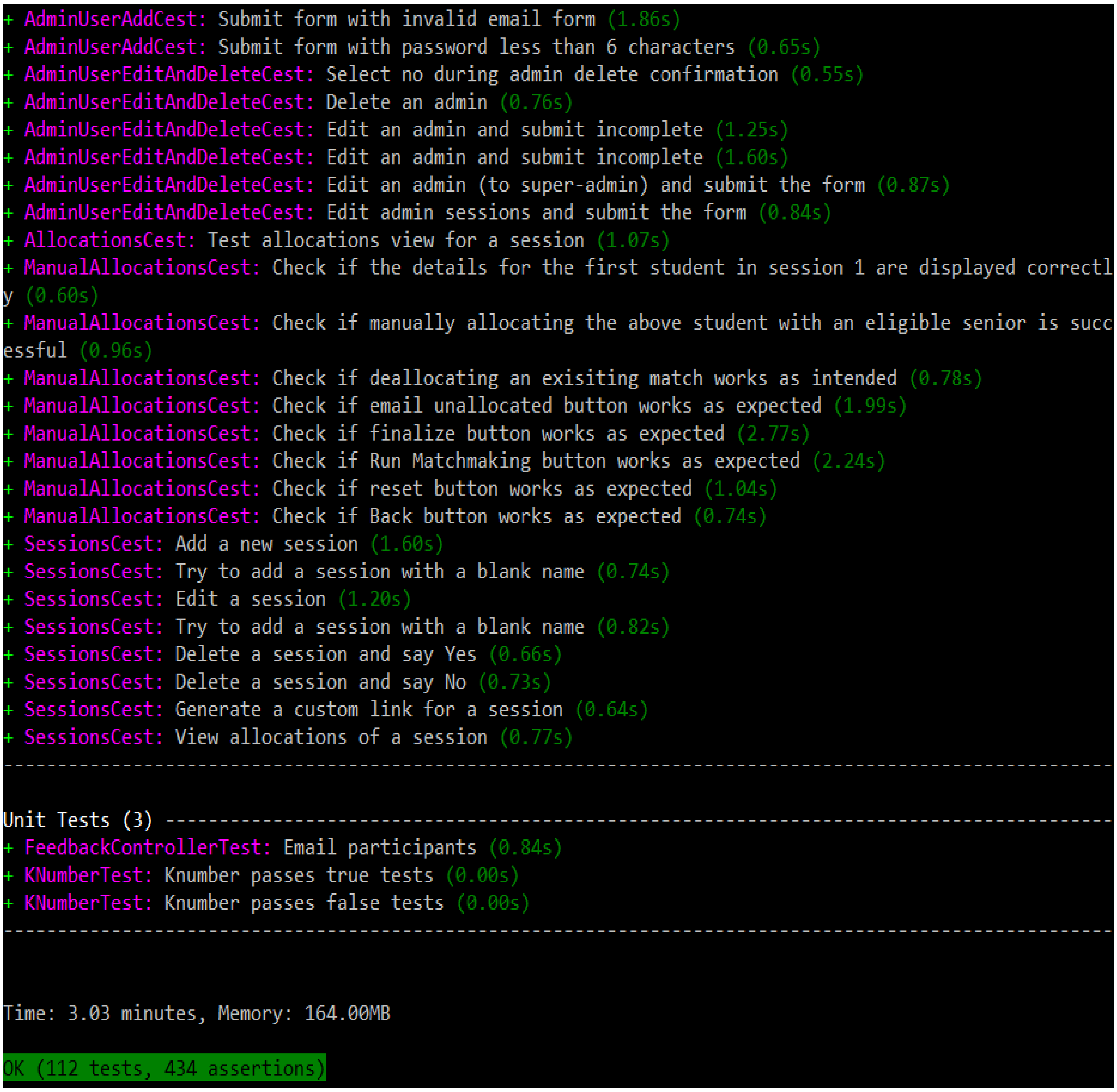
```
Functional Tests (87) --------------------------------------------------------------------------
+ MatchDeallocationCest: Test Finalized Matches Can be Deallocated (1.36s)
+ MatchmakingAlgoCest: Test matching algo matches students correctly (2.59s)
+ MatchmakingAlgoCest: Test When Junior Cant Be Allocated because not enough Senior (0.68s)
+ MatchmakingAlgoCest: Test Junior Allocated With Senior With Which All Interests Match Without Constra
ints (0.78s)
+ MatchmakingAlgoCest: Test When Junior Cant Be Allocated because not enough Senior (0.98s)
+ MatchmakingAlgoCest: Test Junior With Same Gender Allocation Preference is correctly matched with a s
enior of same gender with maximum possible interests choices in common (0.87s)
+ MatchmakingAlgoCest: Test Student With Special Needs Is Allocated A Match Despite Disproportionate nu
mber of juniors and seniors (1.44s)
+ MatchmakingAlgoCest: Test When Disproportionate Number Of Senior And Junior, Student Least Compatible
 With Anyone Else Is Chosen To Be Left Out In Matchmaking (1.00s)
+ MatchmakingAlgoFinalizeCest: Test Finalization Of Matches Work (1.86s)
+ MatchmakingAlgoFinalizeCest: Test that matches are not already finalized after running the matchmakin
g algo (0.97s)
+ MatchmakingAlgoFinalizeCest: Test Finalized Matches Cannot be Deallocated (2.84s)
+ MatchmakingAlgoFinalizeCest: Test Finalized Matches Can be Deallocated (1.24s)
+ MatchmakingAlgoResetCest: Test Matchmaking Algo Reset After Running (1.34s)
+ AdminAccessSecurityCest: Test that an an unauthenticated admin does not have access to super admin ad
min index page (0.57s)
+ StudentAccessSecurityCest: Test that an an unauthenticated student does not have access to admin dash
board (0.48s)
+ StudentAccessSecurityCest: Test that an an unauthenticated student does not have access to admin inte
rest index page (0.94s)
+ StudentAccessSecurityCest: Test that an an unauthenticated student does not have access to super admi
n admin index page (0.55s)
+ UserAccessSecurityCest: Test that an an unauthenticated user does not have access to student area (0.
05s)
+ UserAccessSecurityCest: Test that an an unauthenticated user does not have access to admin dashboard
(0.08s)
+ UserAccessSecurityCest: Test that an an unauthenticated user does not have access to admin interest i
ndex page (0.04s)
+ UserAccessSecurityCest: Test that an an unauthenticated user does not have access to admin session in
dex page (0.07s)
+ UserAccessSecurityCest: Test that an an unauthenticated user does not have access to super admin admi
n index page (0.26s)
+ CustomLinkCest: Successfully create a session and generate link (0.55s)
+ CustomLinkCest: Successfully create a session and generate link with large ID (0.69s)
+ CustomLinkCest: Successfully register through link (0.46s)
+ CustomLinkCest: Successfully opt out of session (0.69s)
```

Functional Tests (Continued)



```
+ CustomLinkCest: Fail generating a link without no session id (0.33s)
+ CustomLinkCest: Fail to register through non-existent ID link (0.56s)
+ CustomLinkCest: Fail to opt out from a session signed up to manually (2.10s)
+ DeleteAccountCest: Delete an account (0.68s)
+ DeleteAccountCest: Select not to delete an account (0.67s)
+ DeleteAccountCest: Fail to delete account (Access /delete_account without logging in) (0.02s)
+ LogoutCest: Logout from an account (0.53s)
+ LogoutCest: Fail to logout (Access /logout without logging in) (0.22s)
+ ProfileCest: Update profile as a buddy (1.13s)
+ ProfileCest: Update profile as a mentor (1.45s)
+ ProfileCest: Add special needs to profile (1.17s)
+ ProfileCest: Add all interests to my profile (1.44s)
+ ProfileCest: Not see any interests when session has none (1.16s)
+ RegisterCest: Successfully register an account (0.42s)
+ RegisterCest: Fail registration (all fields empty) (0.19s)
+ RegisterCest: Fail registration (email does not have @) (0.11s)
+ RegisterCest: Fail registration (passwords do not match) (0.50s)
+ RegisterCest: Fail registration (account already exists) (0.17s)
+ StudentDashboardCest: View student dashboard (0.56s)
+ StudentDashboardCest: Click complete profile link (1.23s)
+ StudentDashboardCest: Click verify k-num link (0.99s)
+ VerifyKNumCest: Successfully send a verification email (1.87s)
+ VerifyKNumCest: Fail to send a verification email (Only letters used in k number) (1.67s)
+ VerifyKNumCest: Fail to send a verification email (Only numbers used in k number) (1.25s)
+ VerifyKNumCest: Fail to send a verification email (Only symbols used in k number) (0.57s)
+ VerifyKNumCest: Fail to send a verification email (k and symbols in k number) (0.73s)
+ VerifyKNumCest: Fail to send a verification email (k number > 8) (0.77s)
+ VerifyKNumCest: Fail to send a verification email (k number < 8) (0.83s)
+ AdminInterestCest: Delete an interest (0.52s)
+ AdminInterestCest: Add and do not delete a visible interest (0.85s)
+ AdminInterestCest: Edit an interest (0.77s)
+ AdminInterestCest: Delete a non-existent interest (0.90s)
+ AdminUserAddCest: Add an admin (0.71s)
+ AdminUserAddCest: Add two admin with same email (1.57s)
+ AdminUserAddCest: Submit incomplete admin register form (1.56s)
+ AdminUserAddCest: Submit form with password and password confirmation fields not matching (0.72s)
```

Functional Tests (continued) & Unit Tests



```
+ AdminUserAddCest: Submit form with invalid email form (1.86s)
+ AdminUserAddCest: Submit form with password less than 6 characters (0.65s)
+ AdminUserEditAndDeleteCest: Select no during admin delete confirmation (0.55s)
+ AdminUserEditAndDeleteCest: Delete an admin (0.76s)
+ AdminUserEditAndDeleteCest: Edit an admin and submit incomplete (1.25s)
+ AdminUserEditAndDeleteCest: Edit an admin and submit incomplete (1.60s)
+ AdminUserEditAndDeleteCest: Edit an admin (to super-admin) and submit the form (0.87s)
+ AdminUserEditAndDeleteCest: Edit admin sessions and submit the form (0.84s)
+ AllocationsCest: Test allocations view for a session (1.07s)
+ ManualAllocationsCest: Check if the details for the first student in session 1 are displayed correctl
y (0.60s)
+ ManualAllocationsCest: Check if manually allocating the above student with an eligible senior is succ
essful (0.96s)
+ ManualAllocationsCest: Check if deallocating an exisiting match works as intended (0.78s)
+ ManualAllocationsCest: Check if email unallocated button works as expected (1.99s)
+ ManualAllocationsCest: Check if finalize button works as expected (2.77s)
+ ManualAllocationsCest: Check if Run Matchmaking button works as expected (2.24s)
+ ManualAllocationsCest: Check if reset button works as expected (1.04s)
+ ManualAllocationsCest: Check if Back button works as expected (0.74s)
+ SessionsCest: Add a new session (1.60s)
+ SessionsCest: Try to add a session with a blank name (0.74s)
+ SessionsCest: Edit a session (1.20s)
+ SessionsCest: Try to add a session with a blank name (0.82s)
+ SessionsCest: Delete a session and say Yes (0.66s)
+ SessionsCest: Delete a session and say No (0.73s)
+ SessionsCest: Generate a custom link for a session (0.64s)
+ SessionsCest: View allocations of a session (0.77s)
--------------------------------------------------------------------------------------------------------

Unit Tests (3) -----------------------------------------------------------------------------------------
+ FeedbackControllerTest: Email participants (0.84s)
+ KNumberTest: Knumber passes true tests (0.00s)
+ KNumberTest: Knumber passes false tests (0.00s)
--------------------------------------------------------------------------------------------------------


Time: 3.03 minutes, Memory: 164.00MB

OK (112 tests, 434 assertions)
```