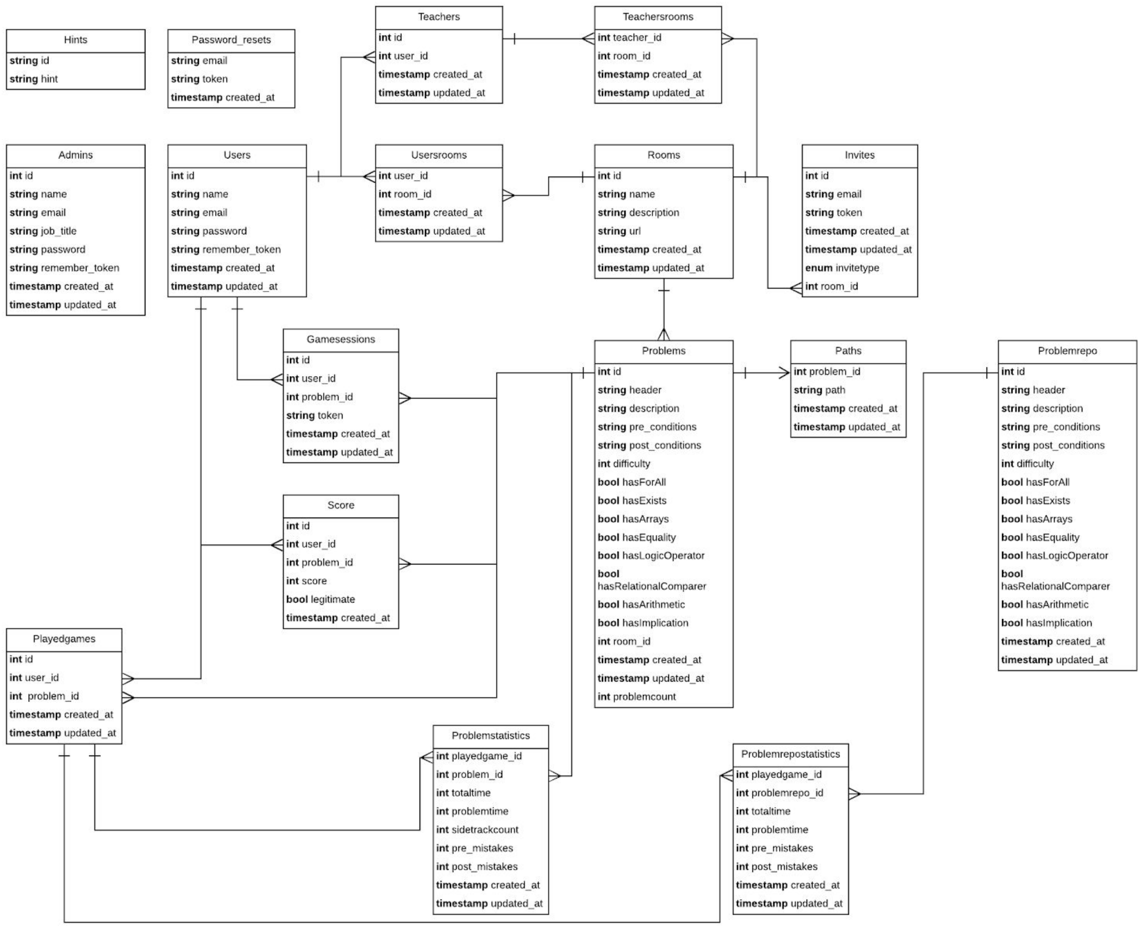
Ludiscite web server documentation

Description

This document provides additional documentation for the web server of the Ludiscite/FormalZ project. The web server is located within the public_html folder of the project. Because Laravel, the framework used for the web server, is built with the model-view-controller-model in mind, the document will be structured around this model.

Database

Below is the entity relationship diagram for the database structure:



The *users* table contains the login information of registered users. Admins are not users, instead their credentials are stored within the *admins* table. A user can be a student of multiple rooms through the *usersrooms* table and can have teacher authentication if there is an entry with that user_id within the *teachers* table. The *teachersrooms* table allows teachers to be the owner of rooms, which are stored in the *rooms* table. Rooms can then contain problems, stored within the *problems* table. Adding a problem to a room will generate a random path for within the game that is then stored in the *paths* table. There is also a problem repository with various problems, contained within the *problemrepo* table. Users can then play problems in their room, which will generate an entry in the *gamesessions* table to notify the back end what user has started a session though a token, sent from the front end, which can be checked against the database entry. The back end will generate entries into the *score* table for their high scores and entries in the *playedgames* table to show that a game has been completed. This will generate statistics for both problems that teachers have added and problems within the problem repository, which are respectively stored in the *problemstatistics* and *problemrepostatistics* tables. The *invites* table stores invites for new students or teachers. Finally there is the uncoupled table *hints,* which stores feedback for different situations and the uncoupled table *password_resets,* which stores entries when a password reset is requested, so that passwords can be reset from an email link.

Laravel allows for the creation of database migrations, which are included within the final product. These serve as a version control of the database structure.
The *Admin, User, Room, Problem, Path, Invite* and *Gamesession* models correspond to the tables with their plural names within the database.

Models

Laravel does not explicitly define what a model within the framework is. This document will refer to the php files that are located within the impress/app folder/namespace as models. In this section there will be documentation of these models. Most models have a similar structure, with a protected *$fillable* and protected *$hidden* array, where $fillable are the fillable attributes (as strings) of the model and *$hidden* are the hidden attributes. Models are coupled to database tables (that are named after the model, but with a trailing s unless defined otherwise) using the Eloquent ORM that is contained within Laravel.

Admin.php

The *Admin* class defines a model for an Administrative user, who is able to log in and invite teachers. It contains fillable attributes *name, email, job_title* and *password* and has hidden attributes *password* and *remember_token*. In addition, it has a protected *$guard* variable specifying the authentication guard for admins.

GameSession.php

The *GameSession* class defines a session within the Phaser game. It contains fillable attributes *user_id* to refer to the user who started the session, *problem_id* to refer to the problem being played and *token* to contain a unique token.

Invite.php

The *Invite* class defines an invite that might be sent by an admin to invite a teacher, or by a teacher to invite a student to a room. It contains fillable attributes *email* to contain the email address that the invite was sent to, *token* to contain a unique token, *invitetype* to define if the invite is a teacher- or roominvite and (optional) *room_id* to refer to the room that the invite is for.

Path.php

The *Path* class defines a path within the Phaser game. It has fillable attributes *problem_id*, to refer to the associated problem and *path* to store a string representation of the path.

Methods

Public static generatePath($problemid)

Generates a path using an external java .jar file stored in impress/resources/pathgeneration/PathGeneration.jar for a certain problem and stores this within the model that is returned. The *$problemid* parameter is the id of the problem that the path should be generated for.

Problem.php

The *Problem* class represents a final problem of an exercise as defined by a teacher. It has fillable attributes *header, description* to define the problem, *pre_conditions* and *post_conditions* to define the conditions, *difficulty* to store the difficulty of the problem, *room_id,* to determine in which room this problem is stored, *hasForAll, hasExists, hasArrays, hasEquality, hasLogicOperator, hasRelationalComparer, hasArithmetic, hasImplication* to classify the problem using different attributes and *problemcount* to store the amount of intermediate problems.

Methods

Public remove()

Deletes the problem and all associated data within the database.

Room.php

The *Room* class defines a classroom that teachers can manage. It has fillable attributes *name, description* to define the room, *url* to store the url for the room and *linkinvite,* to store a boolean defining if the room has email or link invites.

Methods

Public isCreator($userid)

Takes a user id and returns a bool which is true if the user has created this classroom and returns false otherwise.

Public getStatistics($iscreator)

Gets an array of statistics for this problem, with different statistics if the user is the creator of the room. It returns a dictionary of different statistics with attributes:
'completecount' -> the amount of students who have completed the problem
'averagescore' -> average score of the problem
'averagetime' -> average time it took users to complete the problem
'times' -> full array of people's times on the problem
'lastgameproblems' -> statistics of the last game of the user
'completedmails' -> mails of students who have completed the problem.
'Lastgameproblems' is the only attribute that is filled in if *$iscreator* is false.

Public findRepoProblems($problemcount)

Returns a bool classifying if there are enough repoproblems to add the given count of related problems *$problemcount* to this problem.

Public classifyProblems($pre, $post)

Classifies the two strings *$pre* and *$post* and returns a dictionary with the classification attributes ('hasForAll', 'hasExists' etc.) that contain bools if the conditions contain those elements.

Private checkConditionString($pre, $post, $string)

Returns a bool defining if the two strings *$pre* and *$post* contain the substring *$string*.

User.php

The *User* class defines a user, which can be either a teacher or a student. It has fillable attributes *name, email* and password and hidden attributes *password* and *remember_token* which are used in the same way as in the *Admin* class.

Views

Views within Laravel are implemented through blade.php, which provides a templating system. All the views within this project are stored within the impress/resources/views folder. The only layout, the app layout is contained within that folder, within the layouts folder. All the views within the project extend that layout. It provides the top bar and the drop-down menu within the web application. Within this section the different views will be documented as well as the variables that are utilised through their blade.php code.

Admin.blade.php

Defines the view for the admin dashboard.

Home.blade.php

Defines the dashboard for a normal user. This could be both a teacher and a student. The view shows the rooms that the user is a student in or a teacher in. It also provides teachers with a button to create a room.

Variables

$userrooms - The list of rooms that the user is a student in.
$teacherrooms - The list of rooms that the user is a teacher in.
$isteacher - Boolean that is true if the user is a teacher and false otherwise.

Welcome.blade.php

Defines the front page of the website.

About.blade.php

Defines the about page of the website with info on the project.

Auth/Admin-invite.blade.php

Defines the interface for an admin to send an invite to a teacher via email.

Auth/Admin-login.blade.php

Defines the login interface where an admin is able to log in.

Auth/Invite.blade.php

Defines the interface where a teacher can send an invite to a student via email or, when the room has invitations via link, the page shows the invite link to copy.

Variables

$room - The room that the invite page is for.

Auth/Login.blade.php

Defines the login interface for normal users.

Auth/Register.blade.php

Defines the interface where a user/teacher can register to a room or as a teacher.

Variables

$invite - The invite that is linked to this register page. Can be null if the invite is for a room and link based.
$link - A boolean which is true if the room is linkinvite, false if the invite is email only or a teacherinvite.
$room - The room that the register page is for. Can be null if this is a teacher invite.

Auth/Passwords/Change.blade.php

Defines an interface where a user can change their password by providing their old password.

Auth/Passwords/Email.blade.php

Defines an interface where a guest can request a password reset to be sent to their email address.

Auth/Passwords/Reset.blade.php

Defines an interface where a user can set up their new password after clicking on a password reset link in their mail.

Emails/Exception.blade.php

Defines the look of the mail that is sent to the administrator when an exception is handled on the site.

Variables

$content - The error stack trace.

Emails/Invite.blade.php

Defines the look of the mail that is sent upon inviting a teacher or student by mail.

Variables

$invite - The invite that the mail was sent with.

Errors/401.blade.php

Defines the look of the page that is displayed to the user upon a 401 error.

Errors/403.blade.php

Defines the look of the page that is displayed to the user upon a 403 error.

Errors/404.blade.php

Defines the look of the page that is displayed to the user upon a 404 error.

Problem/Profile.blade.php

Defines the look of the main problem page. Contains a button to play the game, a button to go back to the problem's room, a table of highscores, a table of your own scores and a page with statistics of your last finished game. If you are a teacher, it will also contain a button to edit the problem, a table of email addresses of students who have completed the problem and a page with statistics on problems.

Variables

$problem - The problem.
$iscreator - A boolean that is true if the user is the creator of the room that the problem is in.

$scores - An array of objects that represent a score, which each have the username, user id date and score associated with that score. (*$s->name, $s->score, $s->user_id, $s->created_at* if *$s in $scores*)
$statistics - An array with various different statistics:
$statistics['lastgameproblems'] - An array of objects that represent the user's performance on the problems in the last game. Each object has the mistakes made in the preconditions, the mistakes made in the postconditions and the difficulty of the problem. *($p->pre_mistakes, $p->post_mistakes, $p->difficulty* if *$p in $statistics['lastgameproblems']*)
$statistics['completecount'] - An integer which is the amount of times the problem has been completed. Can be null if *$iscreator* is false.
$statistics['averagescore'*] - An integer which is the average score of the scores of this problem. Can be null if *$iscreator is false.
$statistics['averagetime'*] - An integer which is the average time users have taken to complete the final problem of this problem. Can be null if *$iscreator is false.
$statistics['times'*] - Array of objects that contain the 'time' taken to finish the problem for each individual attempt. *($p->problemtime if *$p in $statistics['times']*) Can be null if *$iscreator* is false.
$statistics['completedmails'*] - Array of objects that contain the email addresses of people who have completed the problem. *($p->email if *$p in $statistics['completedmails']*) Can be null if *$iscreator* is false.

Problem/Create.blade.php

Defines an interface for a teacher to create a problem, by defining a header, description, pre- and postconditions, a difficulty and an amount of intermediate problems. Also contains a link to the difficulty examples.

Problem/Edit.blade.php

Defines an interface for a teacher to edit a problem, by defining a header, description, pre- and postconditions, a difficulty and an amount of intermediate problems. Also contains a link to the difficulty examples and a link to the deletion page.

Problem/Delete.blade.php

Defines an interface for a teacher to confirm deleting a problem.

Problem/Examples.blade.php

Defines an interface where a teacher can view different examples of problems, one for each difficulty.

Variables

$examples - An array with an example problem for each difficulty.

Problem/Play.blade.php

Defines an interface where users can play a problem with the Phaser typescript game.

Variables

$token - A token to pass to the game to confirm the gamesession.
$problemroute - A route for the game to return to the problem after play.

Room/Profile.blade.php

Defines the look of the main room page. Contains a list of problems in the room with links to their profile pages. If the user is the creator of the room it also shows a list of users in the room and buttons to edit the room, add problems or invite students.

Variables

$iscreator - A boolean that is true if the user is the creator of the room.
$room - The room.
$problems - An array of the problems that are created in this room.
$users - An array of the email addresses of users that are in this room. *($u->email* if *$u in $users*) Can be empty if *$iscreator* is false.

Room/Create.blade.php

Defines an interface for a teacher to create a room, by defining a name, description and if the room should be mailinvite or linkinvite.

Room/Edit.blade.php

Defines an interface for a teacher to edit a room, by defining a name, description and if the room should be mailinvite or linkinvite. Also contains a link to the deletion page.

Room/Delete.blade.php

Defines an interface for a teacher to confirm deleting a room.

Controllers

The logic of the web server happens within the controller classes. These are located in impress/app/Http/Controllers. The controller classes have different functions that mostly receive HTTP requests from so-called routes (defined within impress/routes/web.php) and return HTTP responses.

Controllers/Controller.php

The base controller that provides controller functionality within Laravel through inheritance.

Controllers/AdminController.php

Controls logic related to the admins.

Methods

Public __construct()

Creates the admin authentication in Laravel's authentication middleware.

Public index()

Is called by a get request on the '/admin' route. Returns the admin.blade.php view.

Controllers/HomeController.php

Controls logic related to the homepage/dashboard.

Methods

Public __construct()

Defines that to use this controller, a user should be logged in within Laravel's authentication middleware.

Public index()

Is called by a get request on the '/' route. Returns the home.blade.php view, and passes it its accompanying variables.

Controllers/AboutController.php

Controls logic related to the about page.

Methods

Public about()

Is called by a get request on the ‘/about’ route. Returns the about.blade.php view.

Controllers/InviteController.php

Controls logic related to the invitation of students/teachers.

Methods

Protected invitevalidator(array $data)

Recieves an array of data containing the index ‘email’ and returns a Laravel Validator object which validates the email so that it can be used within an invite.

Public teacherinvite()

Is called by a get request on the ‘/admin/invite’ route, which can only be called by logged in admins. Returns the auth/admin-invite.blade.php view.

Public roominvite($id)

Is called by a get request on the ‘/room/{id}/invite’ route. The id is the url of the room. Returns the auth/invite.blade.php view with the room identified by the $id variable or a 401 if the user is not the creator of the given room.

Public teacherprocess(Request $request)

Is called by a post request on the ‘/admin/invite’ route, which can only be called by logged in admins. Throws a 422 validationerror if the email is already registered to a user, otherwise returns the result of *$this->process(null, $request)*.

Public roomprocess($id, Request $request)

Is called by a post request on the ‘/room/{id}/invite’ route. The id is the url of the room. Returns a 401 if the user is not the creator of the given room. Returns the room/invite.blade.php view if the room is in linkinvite, because this function should not be called. If the user is already registered to the website, it adds the user to the room and returns with a message that the user has been added. If the user is already in the room, it returns an error that the user is already in the room.

Otherwise, it returns the result of *$this->process($room->id, $request)*, where *$room* is the room defined by the given *$id*.

Private process($id, Request $request)

Takes a room_id *$id*, which can be null in the case of a teacher invite and a post request coming from either the *teacherprocess* or *roomprocess* function. It then creates the invite within the database with the right specifications, sends the email to the invited email address and returns with a success message, or an error if the data cannot be validated by the *invitevalidator*.

Public acceptmail($token)

Is called by a get request on the '/accept/{token}' route. Takes an invite token, finds this invite within the database and returns a 404 if the invite does not exist, otherwise it logs out a user if they are logged in and redirects to the auth/register.blade.php view with the right variables for a mail invite.

Public acceptlink($id)

Is called by a get request on the '/room/{id}/join' route. Takes a room id, finds this room within the database and returns a 404 if the room does not exist, otherwise it logs out a user if they are logged in and redirects to the auth/register.blade.php view with the right variables for a link invite, except for if the room is not linkinvite, then it returns a 401.

Controllers/ProblemExampleController.php

Controls logic related to giving teachers difficulty examples.

Methods

Public __construct()

Sets up the middleware so that the controller is only accessible if the user is logged in.

Public show()

If the user is not a teacher, it returns a 401. Otherwise it gets a random problem of each difficulty from the problem repository in the database and returns the problem/examples.blade.php view with the problems. If there is no problem for a certain difficulty, the controller will send NULL to the view and the view will tell that there are no problems for that difficulty to the user.

Controllers/RoomController.php

Controls all the CRUD logic related to rooms.

Methods

Public __construct()

Sets up the middleware so that the controller is only accessible if the user is logged in.

Protected createvalidator(array $data)

Sets up a validator that validates room $data: it checks if the attribute ‘name’ is a filled-in string of max 255 characters that is unique in the rooms database table and it checks if the attribute ‘description’ is a filled-in string of max 2550 characters.

Protected editvalidator(array $data)

Sets up a validator that validates room edit $data: it checks the same attributes as *createvalidator*, but if the name is not edited it does not have to be unique in the database (because that name is already in the database).

Public create(array $data)

Takes $data, which has attributes ‘name’ and ‘description’ and creates a room if the *createvalidator* passes. If the user is not a teacher, returns a 403.

Public showRoomCreationForm()

Is called by a get request on the ‘/createroom’ route. Returns the room/create.blade.php view. If the user is not a teacher, returns a 403.

Public createRoom(Request $request)

Is called by a post request on the ‘/createroom’ route. Passes the data from the request to *create* and then redirects to the room.

Public show($id)

Is called by a get request on the ‘/room/{id}’ route. Gives the room/profile.blade.php view with the room identified by the given id (where id is equal to the room url), its problems and the iscreator boolean as parameters. If the user is the creator it also gives a list of the users in the room to the view, otherwise this list is empty. If the user is not the room’s creator or not a member of the room it returns a 401.

Public edit($id)

Is called by a get request on the ‘/room/{id}/edit’ route. Gives the room/edit.blade.php view with the room identified by the given id (where id is equal to the room url). If the user is not the room’s creator it returns a 401.

Public update($id, Request $request)

Is called by a post request on the '/room/{id}/edit' route. Edits the room identified by the given id (where id is equal to the room url), with the data given within the request, if the data is validated by the *editvalidator.* If the user is not the room's creator it returns a 403.

Public delete($id)

Is called by a get request on the '/room/{id}/delete' route. Gives the room/delete.blade.php view with the room identified by the given id (where id is equal to the room url). If the user is not the room's creator it returns a 401.

Public destroy($id)

Is called by a post request on the '/room/{id}/delete' route. Deletes the room and associated data such as problems and connections between users and teachers and this room. If the user is not the room's creator it returns a 403.

Controllers/ProblemController.php

Controls all the CRUD logic related to problems.

Methods

Protected createvalidator(array $data)

Sets up a validator that validates room $data: it checks if the attribute 'header' is a filled-in string of max 255 characters, it checks if the attributes 'pre_conditions', 'post_conditions' and 'description' are a filled-in strings of max 2550 characters. Also checks if the attribute 'difficulty' is a filled-in int between 1 and 5 and the attribute 'problemcount' is a filled-in int between 0 and 10.

Public create(array $data)

Is called by a get request on the 'room/{id}/createproblem' route. Returns the problem/create.blade.php view with the room identified by the given id as url as parameter to the view. If the user is not the room owner, returns a 401.

Public store()

Is called by a post request on the 'room/{id}/createproblem' route. Creates a problem with the given data in the *$request* if the data validates properly in the *createvalidator* and redirects to the 'problem.show' route with the newly generated id of the problem. If the user is not the room owner, returns a 403. Also gives a validation error if there are not enough related problems in the repo (*$problem->findRepoProblems($count))*.

Public show($id)

Is called by a get request on the '/profile/{id}' route. Gives the problem/profile.blade.php view with the problem identified by the given id, the room that the problem is in, a bool signifying if the user is the creator, the list of highscores and the statistics acquired by the *$problem->getStatistics($iscreator)* function. If the user is not the room's creator or not a member of the room it returns a 401 (for the room that the problem is in).

Public edit($id)

Is called by a get request on the '/problem/{id}/edit' route. Gives the problem/edit.blade.php view with the problem identified by the given id. If the user is not the room's creator it returns a 401 (of the room that the problem is stored in).

Public update($id, Request $request)

Is called by a post request on the '/problem/{id}/edit' route. Edits the problem identified by the given id, with the data given within the request, if the data is validated by the *createvalidator.* If the user is not the room's creator it returns a 403 (of the room that the problem is stored in). Also gives a validation error if there are not enough related problems in the repo (*$problem->findRepoProblems($count))*.

Public delete($id)

Is called by a get request on the '/problem/{id}/delete' route. Gives the problem/delete.blade.php view with the problem identified by the given id. If the user is not the room's creator it returns a 401 (of the room that the problem is stored in).

Public destroy($id)

Is called by a post request on the '/problem/{id}/delete' route. Deletes the problem and associated data (by calling *$problem->remove()*). If the user is not the room's creator it returns a 403.

Public play($id)

Is called by a get request on the '/problem/{id}/play' route. Creates a gamesession with an unique token. Returns the problem/play.blade.php view with the given token and a route back to the problem. If the user is not the room's creator or not a member of the room it returns a 401 (for the room that the problem is in).

Public asset($id, $filename)

Is called by a get request on the '/problem/{id}/assets/{filename}' route. Redirects the get request to the correct location for a asset in the Phaser game.

Controllers/Auth/AccountManagementController.php

Controls the logic for changing a password.

Methods

Public __construct()

Sets up the middleware so that the controller is only accessible if the user is logged in.

Protected passwordChangeValidator(array $data)

Sets up a validator that validates password change *$data.* It checks if attribute 'password' and 'new_password' are filled-in strings more than 6 characters. (and 'password' should be confirmed).

Protected changePassword(Request $request)

Called by a post request on the 'user/changepassword' route. Gets the data from the *$request,* validates it using the *passwordChangeValidator* and checks if the password is not too common against the list in impress/resources/commonpasswords/CommonPasswords.txt. If this is all correct, it changes the user's password and logs an event for the change.

Protected showChangePasswordForm()

Called by a get request on the 'user/changepassword' route. Shows the auth/passwords/change.blade.php view.

Controllers/Auth/AdminLoginController.php

Controls the logic for logging in an admin.

Methods

Public __construct()

Sets up the middleware so that the controller is only accessible if the user is a guest or an admin.

Public showLoginForm()

Called by a get request on the 'admin/login' route. Returns the auth/admin-login.blade.php view.

Public login(Request $request)

Called by a post request on the ‘admin/login’ route. Attempts to log in the admin and logs this, logs if the login is successful or unsuccessful. Destroys the previous session in the database if it exists to make sure only one person is logged in and then either gives an error if the login failed or redirects to the admin dashboard.

Controllers/Auth/ForgotPasswordController.php

Controls the logic for sending password reset mails.

Methods

Public __construct()

Sets up the middleware so that the controller is only accessible if the user is a guest.

Public sendResetLinkEmail(Request $request)

Is called by the Laravel password reset route. Validates the mail in the *$request* data using Laravel’s validation for forgotten passwords, sends an email if the mail is registered and tells the user that a mail is sent if the email is registered.

Controllers/Auth/ResetPasswordController.php

Controls the logic for resetting passwords. Uses Laravel’s logic mostly.

Variables

$redirectto - protected string that shows where Laravel should redirect to after password reset. Is set to ‘/home’.

Methods

Public __construct()

Sets up the middleware so that the controller is only accessible if the user is a guest.

Controllers/Auth/LoginController.php

Controls the logic for logging in a user. Uses Laravel’s logic mostly.

Variables

$redirectto - protected string that shows where Laravel should redirect to after log in. Is set to '/home'.

Methods

Public __construct()

Sets up the middleware so that the controller is only accessible if the user is a guest or an admin, logs out if the user is logged in.

Public authenticated()

Called after a successful login. Deletes the session stored within the database and saves the new session so that only one session per user can exist and then redirects to the intended location.

Controllers/Auth/RegisterController.php

Controls the logic for registering in a user. Uses Laravel's logic in RegistersUsers.

Variables

$redirectto - protected string that shows where Laravel should redirect to after registration. Is set to '/home'.

Methods

Public __construct()

Sets up the middleware so that the controller is only accessible if the user is a guest.

Protected validator(array $data)

Sets up a validator that validates register *$data.* It checks if attribute 'password' is a filled-in strings more than 6 characters. (and 'password' should be confirmed). It also checks if 'email' is an email that is filled-in and max 255 characters and unique in the users table. It also checks if 'name' is a max 255 string that is filled-in.

Protected create(array $data)

Gets *$data,* validates it using the *validator* and checks if the password is not too common against the list in impress/resources/commonpasswords/CommonPasswords.txt and if the username is not too profane using *$this->checkProfanity().* If this is all correct, it registers the user.

Public registerteacher($token, Request $request)

Is called by a post request on the 'teacher/register/{token}' route. Checks if the invite given by *$token* exists, otherwise returns a 403. Then it creates the teacher using *create($data)* on the dat from the *$request* and logs the registration and redirects.

Public registerusermail($token, Request $request)

Is called by a post request on the 'room/registermail/{token}' route. Checks if the invite given by *$token* exists with the mail given in the data from *$request*, otherwise returns a 403. Then it creates the teacher using *create($data)* on the data from the *$request,* adds them to the correct room and logs the registration and redirects.

Public registeruserlink($id, Request $request)

Is called by a post request on the 'room/{id}/join' route. Checks if the room given by *$id* (which is equal to the url) exists with the linkinvite set to true, otherwise returns a 403. Then it creates the teacher using *create($data)* on the data from the *$request,* adds them to the correct room and logs the registration and redirects. If the user already exists in the room it returns an error to the user.

Public showRegistrationForm()

Returns a 404 to block a standard Laravel route.

Public register()

Returns a 404 to block a standard Laravel route.

Public checkProfanity($name)

Returns true if the string *$name* contains profanity according to the resources/profanity/Profanity.txt file. Returns false otherwise. Uses *profanityRegEx* to check different character combinations.

Public checkProfanity($word)

Gives a string of RegEx to search for combinations for a certain string *$word* of profanity to block out different character combinations. (Example: @@$ instead of ass).

Configuration

The configuration of the web server can be done using a file named *.env* in the *public_html/impress* folder. This file contains the following important variables to configure the web server functionality:

The first section defines the name and environment of the product:
Make sure the URL matches the URL of the website that the product is run on. The app log level defines what amount of event is logged. Make sure to also check the Laravel documentation for this.

```
APP_NAME=Ludiscite
APP_ENV=production
APP_KEY=base64:guiDZq4HpnsMfvypfUGx6cLDHZ/9HfL1e1sa5ZRIj1E=
APP_DEBUG=false
APP_LOG_LEVEL=debug
APP_URL=science-vs160.science.uu.nl:8080
```

The second section defines the database variables. Make sure the variables match the database configuration.

```
DB_CONNECTION=mysql
DB_HOST=localhost
DB_PORT=3306
DB_DATABASE=database_name
DB_USERNAME=root
DB_PASSWORD=password
```

The third section defines where the logs are written to. The Laravel documentation can be checked to see other options.

```
BROADCAST_DRIVER=log
CACHE_DRIVER=file
SESSION_DRIVER=file
SESSION_LIFETIME=120
QUEUE_DRIVER=sync
```

Another important section is the mail section. This defines the email address and mail driver that the email to invite teachers/students is sent from.

```
MAIL_DRIVER=sendmail
```

```
MAIL_HOST=smtp.science.uu.nl
MAIL_PORT=
MAIL_USERNAME=
MAIL_PASSWORD=
MAIL_ENCRYPTION=tls
```

The error mail address variables defines the email address that mails are sent to when a server error is triggered.

```
ERROR_MAIL_ADDRESS=ludiscite@gmail.com
```