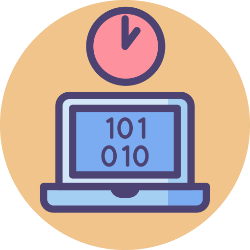
Hackathon management system

Logo



Description

A project created as an international project for UCLL (6 ECTS). This is an IT solution for managing, maintaining and viewing hackathons. Created as an alternative system to the previous one who reportedly was outdated and not sufficient for the needs.

Creators:

* Aurimas Arlauskas

Supervisors:

* Griet Barrezeele
* Krtistien Melaerts
* Jan Goffing

Initial requirements:

Main hackathon page with an overview of the different hackathons.

* Filter options
* Search options

Once you click on a Hackathon you go to the specific hackathon website

* For example how [hackthewaste.com](http://hackthewaste.com/) looks now

Participants to the hackathon can be managed by an admin user

* Participant will get an email to set password
* Participant can maybe edit some details

Participants can be placed in groups by adminParticipants can compete in the hackathon as group

* They can upload various kind of documents
* They can manage and edit their group page

Current workflow:

1. Initial Administrator creates other administrators / evaluators for the Hackathon system. They get their passwords through email.
2. Administrators create Hackathons with a status of “Not started”.
3. During that time, users register to the system. They get email confirmation for successful registration to the system but no logins yet.
4. Administrators gather data and create teams.
5. Administrators assign users to teams.
6. Administrators assign newly created teams to the Hackathon.
7. They start the Hackathon by changing the status to “In progress”. Every participant gets credentials sent to their provided emails.
8. Users sign in into the system and change their credentials(optional).
9. Participants create the Hackathon projects by using a provided text editor and file upload system.
10. Administrator changes the status of hackathon to “ended”. Administrators / evaluators evaluate the projects (not part of the system). Projects become publicly accessible.

Used IT stack:

* React JS for frontend
* C# ASP.NET Web API
* MS SQL Database

Structure:

Structure RTC has three main parts: Frontend, API and Database. Frontend mainly uses React JS, to load data from backend. API for retrieving data from a database, uses C# .Net Framework 4.6 version, combined with Entity Framework. The API was built using Database-First method and to do that it uses ADO.NET data structure to map database tables. MS SQL Database for storing data.

Requirements for deployment:

• C# .Net Framework 4.x version

• Node.JS

• Microsoft SQL Server

• IIS

• Windows 10 or a docker

• Internet Connection

Steps to deploy (Windows server)

1. Upload database file to a local SQL Server (preferably).
2. Add one initial admin user with SSMS or SQL statements.
3. Open Hackathon-backend project with Visual Studio.
4. Update ADO.NET database structure with the new path to the database.
5. Rebuild the project and make sure that all endpoints match without any errors.
6. Build project.
7. Open /hackathon-frontend and open console in that directory.
8. Run “npm install” through console.
9. Open up IIS on Windows server.
10. Provide a new pool for backend’s hosting.
11. Create a new site. Select newly created pool. Provide folder for the built backend project. Mark down the new URL and port.
12. Change all paths that use API requests. Easiest way to do it is with “CTRL + SHIFT + F” and search for localhost.
13. Run “npm build”.
14. Go back to ISS. Create another pool. Create a new site. Provided build folder.
15. Setup SSL certificates.

API Configuration

The following configuration can be found in the Startup.cs class:

1. CORS policy. The project has CORS enabled. There are no set limitations for access to the API.

2. Controllers are tending to maintain the structure of “api/ControllerName/{id}” naming scheme. However, due to the large amount of access points with specific purposes or different privileges most of the Controllers does contain additional link information to specify the request.

API Model

As mentioned above, all of the data is mapped using Entity Framework. To do so, ADO.NET Entity data model is used. The model can be found at “/Models/HackatonDatabaseModel.edmx”. The model is accessed as a private field in each of the controllers.

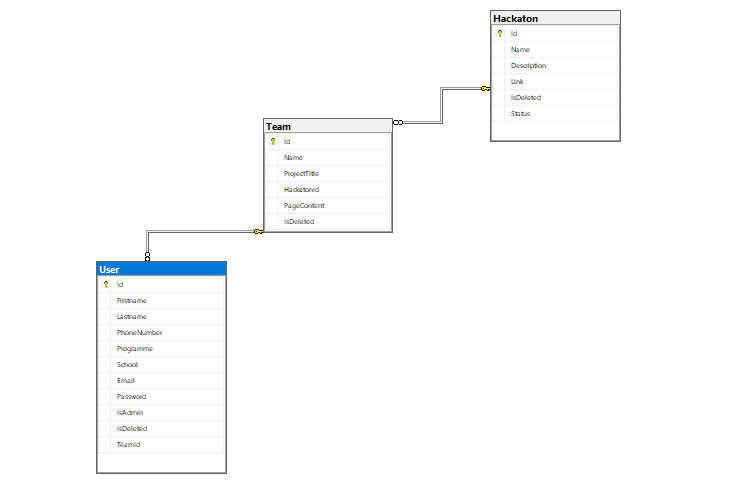
Database

Introduction

This system uses MS SQL Database. This means that is requires Microsoft Database server. Using SSMS is recommended for managing the database. The database consists of 3 database tables:

1. Dbo.Users
2. Dbo.Hackatons
3. Dbo.Teams

Database diagram:



Database tables:

* 1. Hackathon

This is a table for recording all Hackathons.

Fields:

* Id\* (PK)
* Name\* – the title of a Hackathon
* Description – the description for a Hackathon
* Link – link to the website where you can find more information about the Hackathon
* IsDeleted\* - mark field to determine if Hackathon was marked as deleted. Default - false.
* Status\* – has three options. “after” “middle” and “before” each meaning different status of a Hackathon. Default value – “

Relationships:

* 1. Team

Table of a purpose of keeping records of the teams, their projects and file structure.

Fields:

* Id\* (PK)
* Name\* - name of a team
* ProjectTitle\* – Name of the project that the team members pick.
* HackathonId(FK) – The Id of a Hackathon that the Team belongs to.
* PageContent – Contains all of the HTML that the team members upload.
* IsDeleted\* - mark field to determine if Team was marked as deleted. Default - false

Relationships:

OneToMany(hackathonId=> Id(Hackathon)). One hackathon can have many teams.

* 1. User

Table of a purpose of keeping records of the users and their metadata.

Fields:

* Id\* (PK)
* Firstname\*
* Lastname\*
* PhoneNumber
* Programme
* School
* Email\*
* Password\*
* IsAdmin\* - default value - false
* IsDeleted\* - default value - false
* TeamId(FK)

Relationships:

OneToMany(TeamId=> Id(Team)). One Team can have many Users.

\* - this means that the field is required.  
PK – Primary key.  
FK – Foreign key.

API

Introduction

API is written in C# using .NET Web Api framework and Entity framework. The system also manages team project uploaded files. Files are being uploaded in the “/Uploads/{TeamId}” folder. This system also uses email sender which uses SMTP server provided from South Africa’s team.

Each Database table has it’s own separate API controller. Each controller contains endpoints and methods that are logically relevant for that entity.

System is implemented using authorization tokens. The tokens provide each user an identity. Each token has either “user” or “admin” role which is used to protect endpoints from unauthorized usage and provide better structure.

Controllers

* HackathonsController.cs
* TeamsController.cs
* UsersController.cs

HackathonsController

This controller is used for retrieving all of the information that would be entitled to the Hackathon entity.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nr. | Method name | Type | Role | URL | Parameters |
| 1 | GetHackatons | GET | A | /hackatons |  |
| 2 | GetPublicHackatons | GET | P | /hackatons/public |  |
| 3 | GetHackathon | GET | A | /hackatons/{id} | Id - int |
| 4 | GetHackatonTeams | GET | A | /hackatons/teams/{id} | Id- int |
| 5 | GetHackatonTeamsUser | GET | U | /hackatons/teams/user/ |  |
| 6 | GetPublicHackatonTeams | GET | P | /hackatons/public/teams/{id} | Id-int |
| 7 | PutHackaton | PUT | A | /hackatons | Id - int,Hackaton object |
| 8 | AddTeamsToHackaton | PUT | A | /hackatons/add-team/{id} | Id-int, int [] – ids to add |
| 9 | RemoveTeamsFromHackaton | PUT | A | /hackatons/remove-team/{id} | Id-int, int [] – ids to remove |
| 10 | PostHackaton | POST | A | /hackatons | Hackaton object |
| 11 | MarkDeletedHackaton | PUT | A | /hackatons/markdeleted/{id} | Id- int (to mark deleted) |

1. Gets all hackathons that are not deleted. Anonymous object list.
2. Get All hackathons that have status of “after”. Anonymous object list.
3. Gets exact hackathon by id. Anonymous object.
4. Get teams that are part of a hackathon provided by hackathon’s id. Anonymous object.
5. User endpoint for getting all teams that are in the hackathon that he is currently participating at.
6. Get all teams that have participated in an exact hackathon by id. Secure.
7. Updates hackathon information by administrator of a system. Sends email if hackathon starts or ends.
8. Assigns teams to a hackathon. Returns ok();
9. Removes teams from hackathon. Return ok();
10. Adds a new Hackathon with provided information. Returns the created hackathon.
11. Marks hackathon as deleted. Will never be fetched by any of the requests.

UsersController

Used for managing users in the system, relationships, registering new users.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nr. | Method name | Type | Role | URL | Parameters |
| 1 | GetUsers | GET | A | /users |  |
| 2 | GetMyself | GET | A,U | /users/personal |  |
| 3 | GetCredentials | GET | A,U | /users/get-credentials |  |
| 4 | GetUser | GET | A | /users/{id} | Id- int of user |
| 5 | GetUntakenUsers | GET | A | /users/untaken-list/{id} | Id- team’s id that is excluded |
| 6 | PutUser | PUT | A | /users | Id – int, User - object |
| 7 | UpadateCredentials | PUT | A,U | /users/update-credentials | UpdateCredentialRequest |
| 8 | RegisterUserAsync | POST | P | /uses/register | RegisterRequest |
| 9 | MarkDeletedUser | PUT | A | /users/mark-deleted/{id} | Id- int of user |
| 10 | PostUser | POST | A | /users | User object |
| 11 | ResetPasswordAsync | POST | P | /users/reset-password | String-email |

1. Returns anonymous object list of all users that are not marked as deleted.
2. Method used for login. Returns information to keep stored in the localStorage for the sake of displaying necessary user information.
3. Returns user credentials to do their change.
4. Returns an exact user by id. Anonymous object.
5. Returns anonymous list of all users that either have no team or a part of a team specified by an id. Used for assigning users to teams.
6. Updates user information by provided id.
7. Updated only the credentials. Can be done by each user. Specified by token.
8. Api endpoint for registering users to the system as “users”. On a success sends email to the provided email address.
9. Marks user as deleted. Will never again be fetched.
10. Posts a new user into a system by provided information. Returns a new created user with prefiled default values.
11. Endpoint for reseting password by providing an email.

TeamsController

Used for managing teams in the system, relationships, registering new teams, uploading and deleting files.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nr. | Method name | Type | Role | URL | Parameters |
| 1 | GetTeams | GET | A | /teams |  |
| 2 | GetTeam | GET | A | /teams/{id} | Id – team’s id (int) |
| 3 | GetUntakenTeams | GET | A | /teams/untaken-list/{id} | Id – hackathon’s id that is excluded |
| 4 | GetTeamUsers | GET | A,U | /teams/users/{id} | Id – team’s id (int) |
| 5 | PutTeam | PUT | A, U | /teams/{id} | Id – team’s id (int), team-data |
| 6 | AddUsersToTeam | PUT | A | /teams/add-user/{id} | Id-team’s id, int [] – user’s id to add |
| 7 | RemoveUsersFromTeam | PUT | A | /teams/add-user/{id} | Id-team’s id, int [] – user’s id to remove |
| 8 | PostTeam | POST | A | /teams | Team – object |
| 9 | MarkDeletedTeam | PUT | A | /teams/mark-deleted/{id} | Id – team’s id to delete |
| 10 | GetTextAdmin | GET | A | /teams/admin/text/{id} | Id-team’s id |
| 11 | GetPublicText | GET | P | /teams/public/text/{id} | Id – team’s id |
| 12 | GetText | GET | U | /teams/text{id} |  |
| 13 | PostText | POST | U | /teams/text | SiteContentModel object |
| 14 | GetFileList | GET | U | /teams/file-list |  |
| 15 | GetFileListAdmin | GET | A | /teams/admin/file-list/{id} | Id-team’s id |
| 16 | GetFileListPublic | GET | P | /teams/public/file-list/{id} | Id-team’s id |
| 17 | DownloadFile | GET | U | /teams/download-file/{filename} | String filename |
| 18 | DownloadFileAdmin | GET | A | /teams/admin/download-file/{id} | Id-team’s id, String filename |
| 19 | DownloadFilePublic | GET | P | /teams/public/download-file/{id} | Id-team’s id, String filename |
| 20 | UploadFiles | POST | U | /teams/upload-files | Form Content of file |
| 21 | GetDetails | GET | U | /teams/team-details |  |
| 22 | GetDetailsAdmin | GET | A | /teams/admin/team-details/{id} | Id- team’s id |
| 23 | GetPublicDetails | GET | P | /teams/public/team-details/{id} | Id- team’s id |
| 24 | SetDetails | PUT | U | /teams/team-details | TeamDetailsRequest object |

1. Get’s all teams that are not deleted.
2. Get exact team specified by id.
3. Get user details of a team specified by id.
4. Returns all teams that are either untaken or belong to a hackathon specified by id.
5. Put’s a team with provided id and data.
6. Adds teams from a hackathon by provided hackathon’s id and all teams from array of team ids.
7. Removes teams from a hackathon by provided hackathon’s id and all teams from array of team ids.
8. Adds a new team. Returns a newly created team.
9. Marks team as being deleted. Will never be fetched.
10. Get SiteContents as admin by id.
11. Get SiteContents for the team that the belongs to.
12. Gets site contents by provided id. Only returns if the hackathon has been marked as finished.
13. Posts HTML to the site contents. Updates it.
14. Returns a list of file names that belong to a specific project that the user is part of.
15. Returns all file names that belong to a team specified by id.
16. Returns all file names that belong to a team specified by id. Only available for the team that belong to hackathon that is marked as finished.
17. Downloads a file provided by its filename. Only possible for the team that the user is part of.
18. Downloads file of a team specified by id. File is specified by it’s filename provided from a body.
19. Downloads file of a team specified by id. File is specified by it’s filename provided from a body. Only available if the team’s hackathon has ended.
20. Uploads files for the project. Is only available for users that have their team and that team is part of an ongoing hackathon. Provides files as “blob”.
21. Retrieves anonymous object of Team’s name and project title. Only returns for the user that has a team.
22. Retrieves anonymous object of Team’s name and project title. Specified by team’s id.
23. Retrieves anonymous object of Team’s name and project title. For any visitor who wants to see a project. Specified by team’s id. For the team that is part of ended hackathon.
24. Sets new team’s name and team’s title. Only available to user.

Frontend

Introduction

The project’s frontend part was built using react JS and its modules. It uses npm as it’s package manager. Standard CRUD is managed by Redux. The custom parts try to avoid it.

Packages

The following modules and versions are being used in the system. You can find the same specifications in “/package.json”.

Redux

These are the values that are being kept in the Redux initial state:

    Hackatons: null,

    Users: null,

    Teams: null

All of these objects are array based and have CRUD logic provided for them when concerning admin. User or AnonymousUser has no redux functionality, because it is only used for “Read”.

"dependencies": {

    "@material-ui/core": "^4.11.0",

    "@material-ui/icons": "^4.9.1",

    "@material-ui/lab": "^4.0.0-alpha.56",

    "@material-ui/styles": "^4.10.0",

    "@testing-library/jest-dom": "^4.2.4",

    "@testing-library/react": "^9.5.0",

    "@testing-library/user-event": "^7.2.1",

    "antd": "^4.4.2",

    "axios": "^0.21.0",

    "draft-js": "^0.11.7",

    "install": "^0.13.0",

    "material-table": "^1.69.1",

    "material-ui-dropzone": "^3.5.0",

    "node-sass": "^4.14.1",

    "npm": "^6.14.8",

    "qs": "^6.9.4",

    "react": "^17.0.1",

    "react-dom": "^17.0.1",

    "react-image-gallery": "^1.0.8",

    "react-images-upload": "^1.2.8",

    "react-quill": "^1.3.5",

    "react-redux": "^7.2.2",

    "react-router": "^5.2.0",

    "react-router-dom": "^5.2.0",

    "react-router-redux": "^4.0.8",

    "react-rte": "^0.16.3",

    "react-scripts": "3.4.4",

    "react-spinners": "^0.9.0",

    "react-virtualized-transfer": "^1.4.1",

    "redux-thunk": "^2.3.0",

    "request": "^2.88.2"

  },

Structure

* App starts from index.js.
* Index.js renders the router. By default it render’s login screen.
* Router has three different layouts: “admin”, “user” and “public” and it is based on the stored values in localStorage. This is secure, because everything that is in backend is protected by Token Authorization and Authentication.
* Router renders Containers. Each container has it’s logic for providing data for the components.
* Containers are being kept in folder “/Containers”.
* Components are being kept in folder “/Components”;
* API exported functions are kept in 4 different files based on their headers.
* Redux layout can be found in the root folder.
* Images folder contains the images that are being used in this project.

Public containers

These are the locations that can be accessed by any user that has not logged in into the system.

1. LoginDisplay – used for logining in to the system. Uses default route “/”. Redirects to it at a bad route. Provides custom endpoints for retrieving the tokens.
2. RecoverPassword – component used for a form to ask for password recovery. If email is correct it will send out the email to the email that is in the system.   
   Path “/forgot-password”.
3. PublicHackathonsDisplay is used for site visitors that want to see all of the previous hackathons that have been marked as ended.Provides MTable with hackathon’s name, description and a link. On press will redirect to a list of teams.  
   Path: “/previous/hackathons”
4. Teams of a public hackathon that has ended. Provides MTable information on its title, project’s name and number of users. On Click will redirect to viewable project that is considered finished.  
   Path: “/previous/hackathon-details/:id. Id is of a hackathon.
5. Viewable custom object for seeing team’s name, team’s project title, the project’s html and file list(downloadable).  
   Path: “/previous/project-viewer:id”. Id is of a team.
6. RegisterDisplay is used for registering. Will register a new “user” to the system that is not deleted.  
   Path: “/register”

User containers

1. UserTeamsDisplay. Teams of a hackathon that the user is currently a part of. Shows data of project’s name, project’s title and a number of users. OnClick will redirect to a table which show basic information of other team’s users.  
   Path: “/teams”
2. TeamDetailsDisplay shows basic information about each user that is participating in other’s teams.  
   Path: “/team-details:id”. Id is of a team that the user is viewing.
3. ProjectEditorDisplay is used for editing a project, that the user is part of. Provides information from API by making requests about team details, team’s SiteContent and team’s file list. Editable. Is used for creating the project. Using QuillJS.  
   Path: “/project-editor”.
4. ProjectViewer does the same thing as the public one. Difference is that this one is only accessible while the user has access to the system. Main point is to see how the project looks like after the changes have been made.  
   Path: “/project-viewer”. Provides data by the same requests as the editor. By token.
5. CredentialsDisplay is used for changing the personal credentials. Can change Firstname, Surname, Email and Password. Does so by identifying with tokens.  
   Path: “/credentials”.
6. On a wrong url redirects to “/teams”.

Admin containers

1. HackathonsDisplay. Default route. Provides MTable for all hackathons. Has CRUD functionality. By changing the status, the hackathon can be created, started or ended. Also has a button that redirects to a list for managing teams. Once clicked on a hackathon’s row will redirect to a page that shows basic information about team’s that are a part of the hackathon.  
   Path: “/hackatons”.
2. TeamsDisplay is used for providing information on teams. Has CRUD functionality. Has a button for a redirect to manage users. Once clicked, shows basic information about the users that are a part of a team. Also has a button to view the project. Will not work unless the hackathon has been assigned and started or ended.  
   Path: “/teams”.
3. UsersDisplay is used for providing information on users. Has CRUD functionality. The information is detailed, providing, phone number, school and programme so that the admin could create teams based on the user’s input.  
   Path: “/users”.
4. Credentials is used for changing personal data. Same as user.  
   Path: “/credentials”.
5. ManageTeamsDisplay has a transfer list. One side contains all of the teams that have not been assigned. The other one contains the ones, that are already part of a Hackathon. Can add or remove teams from Hackathon.  
   Path: “/manage-teams/id”. Id is of the hackathon.
6. ManageUsersDisplay has a transfer list. One side contains all of the users that have not been assigned. The other one contains the ones, that are already part of a Team. Can add or remove users from Team.  
   Path: “/manage-users/id”. Id is of the team.
7. HackathonDetailsDisplay. Can be accessed from the HackathonsDisplay. Will show all teams basic info that are part of hackathon. Has no CRUD. If clicked on a team will redirect to TeamDeatailsDisplay with the id of the selected Team.  
   Path: “hackaton-details/id” Id is of a hackathon.
8. TeamDetailsDisplay. Can be accessed from the TeamsDisplay. Will show all users basic info that are part of team. Has no CRUD.  
   Path: “team-details/id” Id is of a team.
9. ProjectViewerDisplay is the same one that the user and AnonymousUser get. Can only be viewed during the hackathon that has been started or that has been ended. Has no edit.  
   Path: “/project-viewer/id” Id is of the team.

API requests

This project contains four different API files, that each have an Axios setup on the top. Then, each request is made with provided headers. This simplifies accessing the API and reduces the amount of code that is needed for each request.

Files:

1. API.js. Contains most of the requests. These are requests that are being used by signed in users that require a auth token to be attached to header.
2. DownloadAPI.js Contains the requests that are used for downloading files. It takes in it’s header a “blob” so that could files be downloaded.
3. UploadAPI.js is used for uploading files to the server. Only has one request that is being used by the user.
4. PublicAPI.js is used for all of the requests (except login) for getting data that is publicly available without being signed in.

It is worth mentioning that the LoginLayout has it’s own two requests that are done in sequential order for the sake of providing the flow of auth tokens and then the data.