BALAI PENGEMBANGAN TALENTA INDONESIA
PUSAT PRESTASI NASIONAL
SEKRETARIAT JENDERAL
KEMENTERIAN PENDIDIKAN, KEBUDAYAAN, RISET, DAN TEKNOLOGI

MERDEKA BELAJAR

LKS SMK-DIKSUS

# KISI-KISI

## LOMBA KOMPETENSI SISWA SMK TINGKAT NASIONAL TAHUN 2023

## BIDANG LOMBA

## Teknik Desain Laman

*(Web Technologies)*

MERDEKA BERPRESTASI
Talenta Vokasi Menginspirasi

# MODULE SPEEDTEST

# Contents



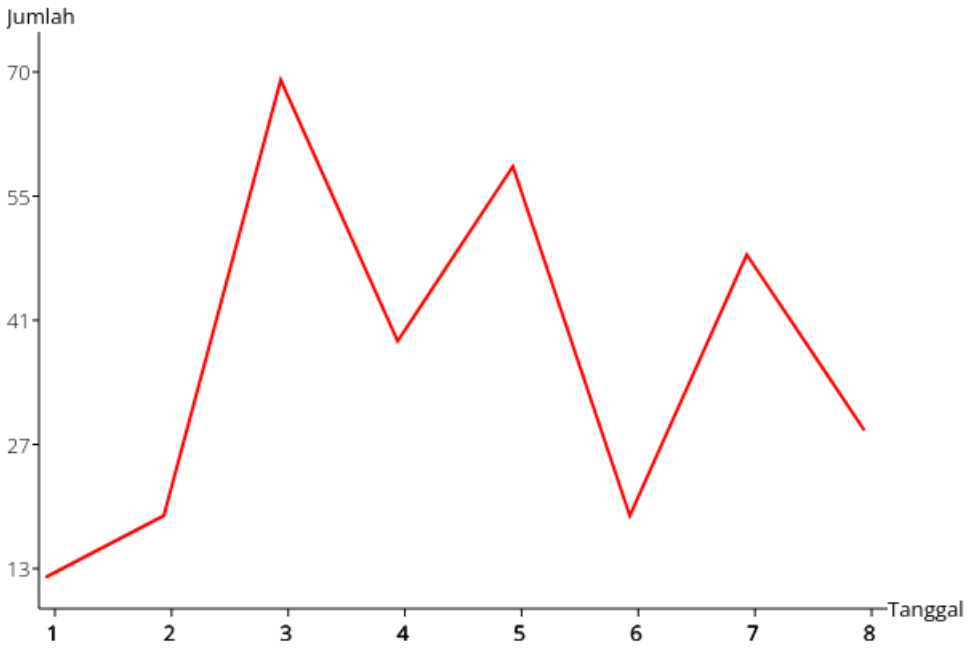
# A1. Landing Page

Create a landing page consisting of navigation bar, logo, menu with dropdown, hero section with heading text, and call to action (CTA).

# B1. Masonry Layout

In a centered container of 960px, you need to make a layout of 3 columns as shown in the image below.

# C1. Line Chart




Make a line chart from the given data provided in media files. The canvas should be 600x400 in size and centered both vertically and horizontally in the browser. The vertical label, horizontal label, and the line should exist within the screen.

# C2. Compare Image

Given an image in the media files, you need to draw the normal image and the grayscale version of the image. Users should drag the circle at the center to left and right in order to see the image difference. Please see the given video example provided in media files.

# C3. Scope

You need to make a hidden fullscreen-sized text, and a box of 400x400 on top of it. The user can not see the text, but they can see it through the box. The box can be moved by users by dragging it. See example.mp4 for more details.

# D1. Internationalization

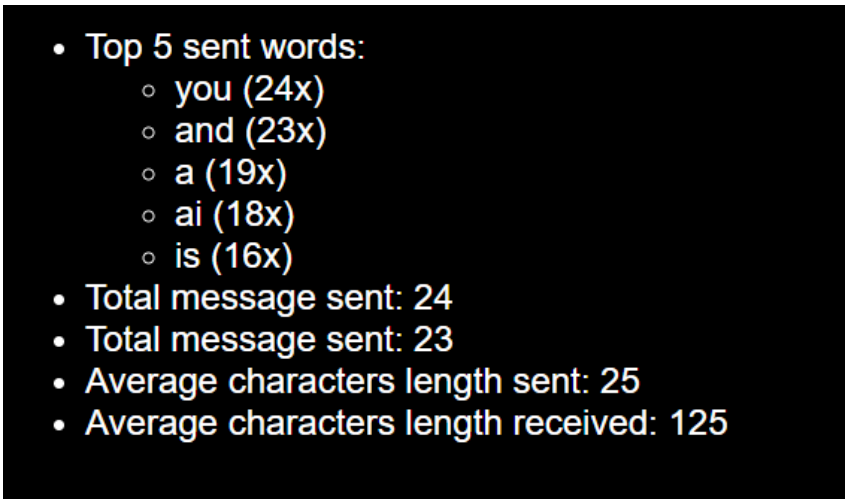
Provided a json file of languages, you need to translate the words that exist in the HTML file using PHP. There is a select form where user can change the language of the website.

# D2. Chat Analytics

Provided a JSON file of user messages, you need to make the analytics of the messages sent which is consist:

- Top 5 sent words
- Total messages sent
- Total messages received
- Average character length sent
- Average character length received

Example:




# D3. Watermark

Given an image and a logo (png file), the web should output an image with a watermark in the top right.

# CMS MODULE

## CONTENTS

**This module has the following files:**

1. MODULE_CMS.doc
2. MODULE_CMS_MEDIA.zip

# Introduction

In 2022, most of the people will prefer to watch movies online instead of going to the cinema. Stream movies online provide many benefits to the user and the streaming service. It is cheap and accessible cross-platform without having to worry about the subscription price.

You are a freelancer working in the field of Web Technologies and you have been asked to develop the brand-new website for anime streaming website called “Animer”. Your client has heard that you are good at building a content management system. They want a theme called “AnimerV1”

# Description of project and tasks

The goal of this website is to present an anime streaming site. It also gives a list of most popular anime. And the goal of this website is to give a streaming service to the visitor and to give a list of articles, which could be of interest to all anime fans. The target audience for this page are the people who like anime. This can be a wide range, therefore you will define your interpretation of the target audience (based on “people who like anime”) and document this definition in your submission. Also please make sure you have the heading title and tagline slogan defined in the content management system.

## Content Management System

For administration tasks, we need two user profiles, Admin and Editor:

- The Admin user - access to the complete WordPress main dashboard.
    - Username: **admin**
    - Password: **admin.**
- The Subscriber user – access as a subscriber role in the CMS.
    - Username: **subscriber**
    - Password: **yourveryfirstsubscriber!**

We want the CMS login page to be white-label. That means the login page should not show the CMS default logo. Thn image of an anime.

Also the login page should not include any "**wp**" wordings for whitee background should be a fullscree-label and security reasons. Please make the backend admin URL as following: **<host>/admeen/**

## Menus

This menu should appear on every page. The client wants to be able to change the menu dynamically via the admin page.

- Home
- Anime List
- Genres (dropdown genre list)
- Season (dropdown season list)
- Articles

## Dashboard Configuration

For simplicity, the client wants to show only some widgets in the dashboard page. Consist of Glance, Activity, Quick Draft.

## Managing the anime

The client wants to be able to create, edit, update, delete anime records. The column that should be shown in every anime is: title, synopsys, release date, rating, and number of episodes.

## Anime Genre

- Action
- Comedy
- Sport
- Romance

## Anime Season

- Winter 2022
- Summer 2022
- Spring 2022
- Fall 2022
- Winter 2023

Users can select **multiple genres and seasons** in an anime.

## URL level of accessibility

The website should have met these URL structures.

- All articles: <host>/articles

- Article detail: <host>/article/<slug>
- All anime: <host>/anime-list
- Anime detail: <host>/anime/<anime slug>
- Episode detail from an anime: <host>/anime/<anime slug>/<episode slug>

## Anime Functionality

In the all anime page, users can choose to sort the anime by view count or anime title, or release date. Users can also choose to filter by the season or genres.

The anime detail page should display title, synopsys, release date, average rating, number of episodes, link to every episode, and a collection of photos (gallery) of the current anime. Admin or editor can display the gallery via shortcode `[anime-gallery id=GALLERY_ID].` View count will be incremented every page visit on an anime detail page.

Users can give their own rating by **clicking one of the five stars**. You need to create the rating functionality.

## Episode Functionality

In every episode of anime, there must be an input for a YouTube video link for the admin to fill. In the episode page, the link should be transformed to an embedded video that the user can play. A link to **go back** to the anime detail page should exist.

## Home page layout and mockup

The layout should be responsive in different screens including mobile and desktop. The home page should consist of three sections.

- Hero section. Displays a big picture of an anime.
- Recommendation section. Shows 4 anime recommendations based on view count.
- Recent anime updates
- Back to top button

## Front-end Requirement

You will need to create your own theme, based on the given starter themes. You will need to implement the following elements for your web page:

- A footer with copyrights and social media links
    - “Copyright © 2022 - All rights reserved” where the year should be the current year dynamically based on server time.
    - 3 social media links. Users are able to change the links for each of the social links.

Feel free to add and change as many elements as you like, but the elements in the list need to be present and your design should blend in with the given design.

## Further optimization tasks

The page needs to be optimized for search engines. You can choose one of the provided plugins or implement your own SEO. Remember to optimize certain parts of your website (e.g. urls, sitemap, …). Secure your page by installing and configuring a security plugin.

Note: If the plugin you would like to choose is broken or does not work as expected, you need to choose another one or feel free to fix this plugin.

## Instructions to the Competitor

The page should follow the Web Content Accessibility Guide.

For the working folder you need to name your folder to "**/CMS_MODULE**".

Dump your database into **"db-dump.sql"** and put it in the wordpress root folder.

# SERVER SIDE MODULE

## Contents

1. MODULE_SERVER_SIDE.docx
2. MODULE_SERVER_SIDE.pdf
3. MODULE_SERVER_SIDE_MEDIA.zip

## Introduction

Your company “Formify Inc.” points you as a Web Developer to build a website which is possible for users to create form dynamically according to the question types such as short answer, paragraph, date input, multiple choice, dropdown and checkboxes. Users who created the form can share the form link to the user to submit a response of the form and also see all of the responses.

## Description of Projects

There are 2 phases in this module. In the first phase, you will make the RESTFul API using Laravel Frameworks according to the provided documentation. In the second phase, you should build a frontend with the Interactive Maps using one of the provided Javascript Frameworks (React/Vue/Angular) and the data must come from the created RESTFul API.

You can find the provided media files to support your work:

- Postman collection and environment
- Api tests (to testing your endpoints automatically)
- Template GUI (to build frontend UI)
- Formify.sql (a database with structures and dummy data)

## Phase 1 : RESTful API

In this phase, you should build a RESTful API using the Laravel framework according to the documentation below.

Ensure users can login using credentials below:

| **Name** | **Email** | **Password** |
|---|---|---|
| User 1 | user1@webtech.id | password1 |
| User 2 | user2@webtech.id | password2 |
| User 3 | user3@worldskills.org | password3 |

# 1. Authentication

You should create Login and Logout endpoints. The **accessToken must be generated by sanctum** and will be placed in the request headers **Authorization Bearer**.

## Login

```
Endpoint     : [DOMAIN]/api/v1/auth/login

Description  : For user to log in the system

Method       : POST

Request      : Body (JSON):
               {
                   "email": "user1@webtech.id",
                   "password": "password1"
               }

Validation   : email:
                  -  required
                  -  valid email address
               password:
                  -  required
                  -  min 5 chars

Response     : If success:
               Headers:
                  -  Response status: 200

               Body (JSON):
               {
                   "message": "Login success",
                   "user": {
                       "name": "User 1",
                       "email": "user1@webtech.id",
                       "accessToken":
"1|QnHFgF0C7ml2LkbDG5QDn34qvJoYGdpxoXy63Poi"
                   }
               }

               If invalid field:
               Headers:
                  -  Response status: 422

               Body (JSON):
               {
                   "message": "Invalid field",
```

```
    "errors": {
        "email": [
            "The email must be a valid email address."
        ],
        "password": [
            "The password field is required."
        ]
    }
}
```

**If failed:**

Headers:
- Response status: 401

Body (JSON):

```
{
    "message": "Email or password incorrect"
}
```

## Logout

Endpoint : [DOMAIN]/api/v1/auth/logout

Description : For user to log out the system

Method : POST

Request : Headers:
- Authorization: “Bearer <accessToken>”

Response : **If success:**

Headers:
- Response status: 200

Body (JSON):

```
{
    "message": "Logout success"
}
```

**If invalid token:**

Headers:
- Response status: 401

Body (JSON):

```
{
    "message": "Unauthenticated."
}
```

# 2. Form

Users can manage (create, see all created forms, and see detail form) with questions to share to invited users based on email domain to submit.

## Create a Form

```
Endpoint     : [DOMAIN]/api/v1/forms

Description  : For user to create new form

Method       : POST

Request      : Headers:
                  -  Authorization: “Bearer <accessToken>”

               Body (JSON):
               {
                 "name": "Stacks of Web Tech Members",
                 "slug": "member-stacks",
                 "allowed_domains": [ "webtech.id" ],
                 "description": "To collect all of favorite stacks",
                 "limit_one_response": true
               }

Validation   : name:
                  -  required
               slug:
                  -  required
                  -  unique
                  -  alphanumeric with special characters only dash “-”
                     and dot “.” and without space
               allowed_domains:
                  -  array

Response     : If success:
               Headers:
                  -  Response status: 200

               Body (JSON):
               {
                   "message": "Create form success",
                   "form": {
                       "name": "Stacks of Web Tech Members",
                       "slug": "member-stacks",
                       "description": "To collect all of favorite..",
                       "limit_one_response": true,
                       "creator_id": 1,
                       "id": 3
                   }
```

```
}

If invalid field:
Headers:
   -  Response status: 422

Body (JSON):
{
    "message": "Invalid field",
    "errors": {
        "name": [
            "The name field is required."
        ],
        "slug": [
            "The slug has already been taken."
        ],
        "allowed_domains": [
            "The allowed domains must be an array."
        ]
    }
}

If invalid token:
Headers:
   -  Response status: 401

Body (JSON)
{
    "message": "Unauthenticated."
}
```

## Get all Forms

```
Endpoint     :  [DOMAIN]/api/v1/forms

Description  :  For users to get all of their created forms

Method       :  GET

Request      :  Headers:
                   -  Authorization: “Bearer <accessToken>”

Response     :  If success:
                Headers:
                   -  Response status: 200
```

Body (JSON):


```
{
    "message": "Get all forms success",
    "forms": [
        {
            "id": 1,
            "name": "Biodata - Web Tech Members",
            "slug": "biodata",
            "description": "To save web tech membe..",
            "limit_one_response": 1,
            "creator_id": 1
        },
        {
            "id": 2,
            "name": "HTML and CSS Skills - Quiz",
            "slug": "htmlcss-quiz",
            "description": "Fundamental web tests",
            "limit_one_response": 1,
            "creator_id": 1
        },
        {
            "id": 3,
            "name": "Stacks of Web Tech Members",
            "slug": "member-stacks",
            "description": "To collect all of favorit..",
            "limit_one_response": 1,
            "creator_id": 1
        }
    ]
}
```


**If invalid token:**

Headers:

- Response status: 401

Body (JSON):


```
{
    "message": "Unauthenticated."
}
```


## Detail Form

Endpoint : [DOMAIN]/api/v1/forms/<form_slug>

Description : For invited users to get detail form

Method : GET

Request : -

Response : **If success:**

Headers:
- Response status: 200

Body (JSON):


```
{
    "message": "Get form success",
    "form": {
        "id": 1,
        "name": "Biodata - Web Tech Members",
        "slug": "biodata",
        "description": "To save web tech members data",
        "limit_one_response": 1,
        "creator_id": 1,
        "allowed_domains": [
            "webtech.id"
        ],
        "questions": [
            {
                "id": 1,
                "form_id": 1,
                "name": "Name",
                "choice_type": "short answer",
                "choices": null,
                "is_required": 1
            },
            {
                "id": 2,
                "form_id": 1,
                "name": "Address",
                "choice_type": "paragraph",
                "choices": null,
                "is_required": 0
            },
            {
                "id": 3,
                "form_id": 1,
                "name": "Born Date",
                "choice_type": "date",
                "choices": null,
```

```
                "is_required": 1
            },
            {
                "id": 4,
                "form_id": 1,
                "name": "Sex",
                "choice_type": "multiple choice",
                "choices": "Male,Female",
                "is_required": 1
            }
        ]
    }
}
```

**If form slug invalid:**

Headers:

- Response status: 404

Body (JSON):

```
{
    "message": "Form not found"
}
```

**If user email domain not allowed to submit form:**

Headers:

- Response status: 403

Body (JSON):

```
{
    "message": "Forbidden access"
}
```

**If invalid token:**

Headers:

- Response status: 401

Body (JSON):

```
{
    "message": "Unauthenticated."
}
```

# 3. Question

Users can manage questions (add and remove) from one of their created forms. Choices field must be required if the user selects multiple choice, dropdown, or checkboxes. There are 6 choice types:

1. Short answer (TextField)
2. Paragraph (TextArea)
3. Date (Input Date)
4. Multiple Choice (Radio) : with choices
5. Dropdown (Select) : with choices
6. Checkboxes (Checkboxes) : with choices

## Add a Question

```
Endpoint     :  [DOMAIN]/api/v1/forms/<form_slug>/questions

Description  :  For user to add a question of a form

Method       :  POST

Request      :  Headers:
                   -  Authorization: “Bearer <accessToken>”

                Body (JSON):
                {
                  "name": "Most Favorite JS Framework",
                  "choice_type": "multiple choice",
                  "choices": [
                    "React JS",
                    "Vue JS",
                    "Angular JS",
                    "Svelte"
                  ],
                  "is_required": true
                }

Validation   :  name:
                   -  required
                choice_type:
                   -  required
                   -  only: “short answer”, “paragraph”, “date”,
                      “multiple choice”, “dropdown”, or “checkboxes”
                choices
                   -  required if selected choice type is “multiple
                      choice”, “dropdown”, or “checkboxes”

Response     :  If success:
                Headers:
                   -  Response status: 200

                Body (JSON):
```

```
{
    "message": "Add question success",
    "question": {
        "name": "Most Favorite JS Framework",
        "choice_type": "multiple choice",
        "is_required": true,
        "choices": "React JS,Vue JS,Angular JS,Svelte",
        "form_id": 3,
        "id": 10
    }
}
```

**If invalid field:**

Headers:

- Response status: 422

Body (JSON):

```
{
    "message": "Invalid field",
    "errors": {
        "name": [
            "The name field is required."
        ],
        "choice_type": [
            "The choice type field is required."
        ],
        "choices": [
            "The choices must be an array.",
            "The choices must be a string."
        ]
    }
}
```

**If invalid form slug:**

Headers:

- Response status: 404

Body (JSON):

```
{
    "message": "Form not found"
}
```

**If try access another user form:**

Headers:

- Response status: 403

Body (JSON):

```
{
    "message": "Forbidden access"
}
```

**If invalid token:**

Headers:
- Response status: 401

Body (JSON):

```
{
    "message": "Unauthenticated."
}
```

## Remove a Question

Endpoint : [DOMAIN]/api/v1/forms/<form_slug>/questions/<question_id>

Description : For user to remove a question of a form

Method : DELETE

Request : Headers:
- Authorization: “Bearer <accessToken>”

Response : **If success:**

Headers:
- Response status: 200

Body (JSON):

```
{
    "message": "Remove question success"
}
```

**If invalid form slug:**

Headers:
- Response status: 404

Body (JSON):

```
{
    "message": "Form not found"
}
```

**If invalid question id:**

Headers:
- Response status: 404

```
Body (JSON):
{
    "message": "Question not found"
}
```

**If try access another user form:**

```
Headers:
   -  Response status: 403

Body (JSON):
{
    "message": "Forbidden access"
}
```

**If invalid token:**

```
Headers:
   -  Response status: 401

Body (JSON):
{
    "message": "Unauthenticated."
}
```

# 4. Response

If the creator form sets the form with limit 1 response, then the invited user can't submit a response twice. Submit Response only can be accessed by users with an allowed domain of the user email. If the creator form does not fill allowed domains, it means that the form can be accessed by anyone/public.

**For example:**
An user with email "user3@worldskills.org" can't submit a response to the form with allowed domains webtech.id and inaskills.id, because worldskills.org can't be allowed to access the form.

## Submit Response

```
Endpoint     :  [DOMAIN]/api/v1/forms/<form_slug>/responses

Description  :  For invited user to send response of a form

Method       :  POST

Request      :  Headers:
                   -  Authorization: "Bearer <accessToken>"
```

Body (JSON):

```
{
    "answers": [
        {
            "question_id": 1,
            "value": "Ica Amalia"
        },
        {
            "question_id": 2,
            "value": "Bandung"
        },
        {
            "question_id": 3,
            "value": "2006-08-01"
        },
        {
            "question_id": 4,
            "value": "Female"
        }
    ]
}
```

Validation : answers
- array
[question value]
- required if sets true

Response : **If success:**

Headers:
- Response status: 200

Body (JSON):

```
{
    "message": "Submit response success"
}
```

**If invalid field:**

Headers:
- Response status: 422

Body (JSON):

```
{
    "message": "Invalid field",
    "errors": {
        "answers": [
            "The answers field is required."
        ]
    }
```

```
}

If user email domain not allowed to submit form:
Headers:
   -  Response status: 403

Body (JSON):
{
    "message": "Forbidden access"
}

If limited for 1 response then user submit twice:
Headers:
   -  Response status: 422

Body (JSON):
{
    "message": "You can not submit form twice"
}

If invalid token
Headers:
   -  Response status: 401

Body (JSON):
{
    "message": "Unauthenticated."
}
```

## Get all Responses

```
Endpoint     :  [DOMAIN]/api/v1/forms/<form_slug>/responses

Description  :  For the creator see all responses

Method       :  GET

Request      :  Headers:
                   -  Authorization: “Bearer <accessToken>”

Response     :  If success:
                Headers:
                   -  Response status: 200

                Body (JSON):
                {
```

```
    "message": "Get responses success",
    "responses": [
        {
            "date": "2022-10-24 01:47:14",
            "user": {
                "id": 1,
                "name": "User 1",
                "email": "user1@webtech.id",
                "email_verified_at": null
            },
            "answers": {
                "Name": "Budi Setiawan",
                "Address": "Jakarta, Indonesia",
                "Born Date": "2004-05-05",
                "Sex": "Male"
            }
        },
        {
            "date": "2022-10-24 02:03:27",
            "user": {
                "id": 2,
                "name": "User 2",
                "email": "user2@webtech.id",
                "email_verified_at": null
            },
            "answers": {
                "Name": "Ica Amalia",
                "Address": "Bandung",
                "Born Date": "2006-08-01",
                "Sex": "Female"
            }
        }
    ]
}
```


**If form slug invalid:**

Headers:

- Response status: 404

Body (JSON):


```
{
    "message": "Form not found"
}
```

**If try access another user form:**

```
Headers:
   -  Response status: 403

Body (JSON)
{
    "message": "Forbidden access"
}
```

**If invalid token**

```
Headers
   -  Response status: 401

Body (JSON)
{
    "message": "Unauthenticated."
}
```

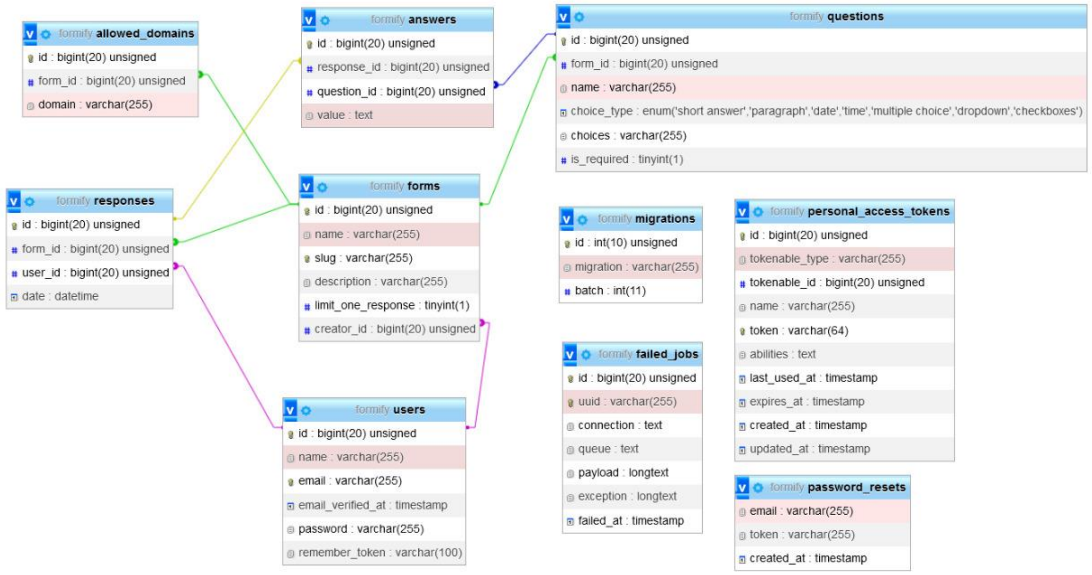
# Phase 2 : Front-end Development

In this part, you should build a front-end application using one of the provided frameworks (vue js or react js). You can use the provided template gui on the media files to build the front-end ui.

| Pages / Features | Description |
|---|---|
| Login and Logout | - User can login using the correct credentials (username and password)<br>- Alert errors should be displayed when login failed<br>- Users can log out by clicking the logout button on the navbar. |
| Home | - Users can see their list of created forms. |
| Create Form | - Form inputs displayed correctly (name, slug, description, allowed domains and limit to 1 response switch). Name and slug fields are required.<br>- Users can create a form by filling in all of the required fields.<br>- Alert errors should be displayed when failed. |
| Detail Form | **General:**<br>- Form detail displayed correctly<br>- Form link displayed correctly<br>- User can copy form link<br><br>**Questions:**<br>- Questions displayed correctly<br>- Question inputs should be disabled when has created<br>- Users can add questions by filling in a question form and clicking the save button.<br>- Choices input should be displayed when the user selects multiple choice, dropdown, or checkboxes.<br>- Alert errors should be displayed when created failed.<br>- Users can remove questions by clicking the remove button.<br><br>**Responses:**<br>- Table displayed correctly according to questions and responses.<br>- Total responses count is displayed correctly. |

| | |
|---|---|
| Submit Form | - Form detail displayed correctly (name, description, and user email).<br>- Questions with the form displayed correctly.<br>- Users can submit by filling in all of the required forms and clicking the submit button.<br>- Submit button should be disabled if there is a required form that hasn't been filled.<br>- Users can't submit twice if the form is limited for 1 response.<br>- Forbidden access page should be displayed when the user isn't invited to access that form. |

# ERD




# Instructions

- Save your REST Api in the directory called **"SERVER_MODULE/backend"** and save it in the repository folder.
- Save your front-end app in the directory called **"SERVER_MODULE/frontend"** and save it in the repository folder.
- When developing the Frontend, you can **use/consume the provided REST API**.
- You should **build the frontend app to production mode** and **save the production files in the "SERVER_MODULE/frontend/build" folder** before you push it to the github repository.
- Make sure the **"/build"** folder **isn't written on the .gitgnore file**.
- You should commit and push your changes to the github repository at least every 30 minutes.

# MODULE CLIENT

## Contents

1. MODULE_CLIENT.docx
2. MODULE_CLIENT_MEDIA.zip

## Introduction

People love to play challenging games which require the player's effort to think. Games will be more challenging if there are numbers involved. Nevertheless, a multiplayer game can be more fun if people play it together.

With the support of current sophisticated technology, we can make a game that can be played in a web browser so people can play it across devices. You need to make this game run correctly in the latest version of Google Chrome or Mozilla Firefox.

## Description of the project

You need to create a game called Hexaria, a fun multiplayer math puzzle game where the objective is to obtain the most points in a hexagonal board during a game.

This is a 5 hours module in which you will need to create the initial layout of the game and the game functionality using Javascript. You are required to make the game work perfectly in modern browsers without any error.

Hexaria game screen should have these components below:

1. Game title
2. 10x8 hexagon blocks
3. Both players names
4. Both players scores
5. Current hexagon
6. New game button
7. Leaderboard
8. Sort Functionality

## Game functionalities

1. **Show the game** icon as a favicon.
2. **Welcome screen should show** when the page is loaded.
3. **Game instruction should appear** besides the welcome screen.
4. **The game instruction** should appear in an animated way.

5. **Player 1 can choose to compete with** Player 2 or Bot.
6. **Player 1 can start the game** by clicking the start button.
7. **If the user chooses to compete with Player 2,** Player 1 must input the Player 1's name and Player 2's name.
8. **If the user chooses to compete with Bot,** Player 1 doesn't need to input bot's name.
9. **Players 1 can choose a level** in the welcome screen (easy, medium, hard).
10. **The "Play Game" button should be disabled** if the user hasn't filled in their name and hasn't chosen a level.
11. **The game board consists** of a 10x8 hexagon block and Every hexagon position in the row should have connected to the above and the bottom row.
12. **Disabled hexagon will appear in the random hexagon with grayish-color:**
    a. **4 disabled hexagon** for easy level
    b. **6 disabled hexagon** for medium level
    c. **8 disabled hexagon** for hard level
13. **Player 1's color** is red and the other is blue.
14. **Red takes the first turn.** Blue takes turns with red. Player 1 or 2 by clicking an empty hexagon and bot by automatically filling an empty hexagon.
15. **Each turn, a hexagon with a random number (1-20)** can be used by either player to be placed in one of the empty hexagons.
16. **The current turn hexagon should be displayed on the screen** to tell the player what color and number of the hexagon that they will put.
17. **When the player hover one of the empty hexagons,** there will be a placeholder of the current hexagon depending on the current hexagon color and number.
18. **When an empty hexagon is chosen by players,** it will be filled with the current turn hexagon.
19. **Player can take over the opponent's hexagons** if the player placed their hexagon in an adjacent opponent's hexagon with a higher value hexagon than the opponent's hexagon.
20. **Taken over hexagon color** will change based on which player's turn.
21. **When a player places a new hexagon adjacent to an identical color hexagon**, the adjacent hexagon will be added up by 1 number.
22. **When an empty hexagon is hovered by a player or bot,** display moves animation which show what will happen to the adjacent hexagon by showing color changing animation or add up 1 animation.
23. **Bot should imitate human moves by showing 3 steps** before its actual step for at least 1 second. Show any moves animation if a step should show animation.
24. **The score will be calculated by sum of every identical color hexagon** that has been placed by players.
25. **Disabled hexagons** can not be hovered or clicked.
26. **Play the sound effect** given in the media files when the player is placing a new hexagon.
27. **The current score of the red and blue player** should be shown on the screen.
28. **When there is no empty hexagon left,** the game is over.
29. **When the game is over, show a popup** to display the name, score of the winner, and the restart button.

30. **Score will be saved** and displayed in the leaderboard and see the details of the player's game by clicking the details button.
31. **Scores data** should persist in the browser.
32. **Players can sort the leaderboard** by the winner score or by the date in descending order.
33. **The game should be working correctly** in Google Chrome and Mozilla Firefox by opening the index.html file directly.
34. **The game should work** without JavaScript errors and messages shown in the browser console

# INSTRUCTION FOR COMPETITORS

1. The media files are available in the ZIP file. You can modify the supplied files and create new media files to ensure the correct functionality and improve the application.
2. The entry file should be **'CLIENT_MODULE/index.html'**
3. You should create additional images for each of the requested resolutions to highlight hidden elements, animations, interactions, or any additional information that will assist in the presentation of the game design.
4. Save the working game to the directory on the server named "**CLIENT_MODULE**". Be sure that your main file is called **index.html**.
5. You are responsible for the time management in your development. If you finalize some tasks you can continue to other tasks.