

**CSCB634 Практика по програмиране и по реализация на бази данни**

**Приложение „Електронен дневник“**

*София, 2024*

**Table of Contents**

**1. Introduction**

**2. Prerequisities**

**3. Getting Started**

* **\*Project Setup**
* **\*Running the Application**

**4. Configuration**

**5. Usage**

* **\*API Endpoints**
* **\*Database Interaction**

**6. Testing**

**7. User manual**

## Introduction

The Web-Based Electronic Diary is designed to facilitate the management of student information for schools, covering grades 1 through 12. This system allows for the input, editing, deletion, and visualization of student results. It offers tailored functionalities for various user roles, including administrators, principals, teachers, parents, and students, ensuring a comprehensive and user-friendly interface for all stakeholders involved in the educational process.

### **Key Features:**

* Role-based access control (administrator, principal, teacher, parent, student)
* User registration and login
* Customizable user interface for different roles
* School data management (name and address)
* Student enrollment and class assignment
* Management of personal data for principals, teachers, students, and parents
* Creation and management of the academic schedule
* Grade and attendance management by teachers
* Parental access to children's grades and attendance
* Principal access to comprehensive school data and statistics
* Administrator control over user data, roles, and system-wide statistics

**Before you begin using the Application, please ensure that you meet the following prerequisites:**

### Software:

* Web Server: Apache or Nginx
* Database Server: MySQL or PostgreSQL
* Development Environment: Node.js and npm
* Programming Language: PHP (back-end development), JavaScript (front-end development)
* Framework: Symfony (back-end), React.js (front-end)
* Additional Libraries: Redux (state management), Bootstrap (responsive design)
* Version Control: Git
* Browser: Latest versions of Chrome, Mozilla Firefox, Internet Explorer

## **Configuration**

* **Database Configuration**: You may need to configure the database connection details in the application's application.properties or application.yml file.
* **Logging**: Configure logging settings to capture application logs effectively.

## Getting Started

Follow these steps to set up and run the Web-Based Electronic Diary:

### Setting Up the Development Environment

1. **Install Node.js and npm:** Download and install the latest version of Node.js from [Node.js official website](https://nodejs.org/). npm is included with Node.js.
2. **Install MySQL/PostgreSQL:** Download and install the preferred database server:
   * [MySQL](https://www.mysql.com/downloads/)
   * [PostgreSQL](https://www.postgresql.org/download/)
3. **Install PHP and Composer:** Download and install PHP from [PHP official website](https://www.php.net/downloads) and Composer from [Composer official website](https://getcomposer.org/).
4. **Set Up the Database:** Create a new database for the application.

**[sql]**

CREATE DATABASE electronic\_diary;

1. **Clone the Repository:** Clone the project repository from GitHub.

**[bash]**

git clone https://github.com/Thurnos/CSCB634.git

cd electronic-diary

### Configuring the Application

1. **Install Dependencies:** Navigate to the project directory and install the necessary dependencies for both the back-end and front-end.

**[bash]**

composer install

npm install

1. **Configure Environment Variables:** Create a .env file in the root directory and add the necessary configuration details.

**[env]**

DB\_HOST=localhost

DB\_USER=root

DB\_PASS=yourpassword

DB\_NAME=electronic\_diary

JWT\_SECRET=your\_secret\_key

### Running the Application

1. **Database Migration:** Run the database migration scripts to set up the necessary tables

**[bash]**

php bin/console doctrine:migrations:migrate

1. **Start the Development Servers:** Launch the Symfony server for the back-end and the React development server for the front-end..

**[bash]**

symfony server:start

npm start

1. **Access the Application:** Open your browser and navigate to http://localhost:3000 to access the web-based electronic diary.

### Controllers and Entities Documentation

For each class in our system, we are using a combination of a Controller and an Entity to manage and interact with the data. Below is a detailed documentation for each controller and its corresponding entity, demonstrating how they work together to perform CRUD (Create, Read, Update, Delete) operations.

#### AttendanceController and Attendance Entity

**Controller: AttendanceController**

* **Namespace:** App\Controller
* **Dependencies:**
  + EntityManagerInterface for managing database operations.
  + AttendanceRepository for interacting with the attendance records.
* **Endpoints:**
  + /attendance/list to list all attendance records.
  + /attendance/get/{id} to retrieve a specific attendance record.
  + /attendance/add to add a new attendance record.
  + /attendance/edit/{id} to update an existing attendance record.
  + /attendance/delete/{id} to delete an attendance record.

**Entity: Attendance**

* **Namespace:** App\Entity
* **Fields:**
  + id: Unique identifier for the attendance record.
  + date: The date of the attendance.
  + status: The attendance status (e.g., present, absent).
  + student\_id: The ID of the student.
  + subject\_id: The ID of the subject.

#### GradesController and Grades Entity

**Controller: GradesController**

* **Namespace:** App\Controller
* **Dependencies:**
  + EntityManagerInterface for managing database operations.
  + GradesRepository for interacting with the grades records.
* **Endpoints:**
  + /grades/list to list all grades records.
  + /grades/get/{id} to retrieve a specific grades record.
  + /grades/add to add a new grades record.
  + /grades/edit/{id} to update an existing grades record.
  + /grades/delete/{id} to delete a grades record.

**Entity: Grades**

* **Namespace:** App\Entity
* **Fields:**
  + id: Unique identifier for the grades record.
  + grade: The grade awarded.
  + student\_ids: The IDs of the students receiving the grade.
  + school\_id: The ID of the school.

#### MarksController and Marks Entity

**Controller: MarksController**

* **Namespace:** App\Controller
* **Dependencies:**
  + EntityManagerInterface for managing database operations.
  + MarksRepository for interacting with the marks records.
* **Endpoints:**
  + /marks/list to list all marks records.
  + /marks/get/{id} to retrieve a specific marks record.
  + /marks/add to add a new marks record.
  + /marks/edit/{id} to update an existing marks record.
  + /marks/delete/{id} to delete a marks record.

**Entity: Marks**

* **Namespace:** App\Entity
* **Fields:**
  + id: Unique identifier for the marks record.
  + mark: The mark awarded.
  + date: The date the mark was awarded.
  + student\_id: The ID of the student.
  + subject\_id: The ID of the subject.

#### ParentsController and Parents Entity

**Controller: ParentsController**

* **Namespace:** App\Controller
* **Dependencies:**
  + EntityManagerInterface for managing database operations.
  + ParentsRepository for interacting with the parents records.
* **Endpoints:**
  + /parents/list to list all parents records.
  + /parents/get/{id} to retrieve a specific parents record.
  + /parents/add to add a new parents record.
  + /parents/edit/{id} to update an existing parents record.
  + /parents/delete/{id} to delete a parents record.

**Entity: Parents**

* **Namespace:** App\Entity
* **Fields:**
  + id: Unique identifier for the parents record.
  + name: The name of the parent.
  + email: The email of the parent.
  + number: The contact number of the parent.

#### PrincipalsController and Principals Entity

**Controller: PrincipalsController**

* **Namespace:** App\Controller
* **Dependencies:**
  + EntityManagerInterface for managing database operations.
  + PrincipalsRepository for interacting with the principals records.
* **Endpoints:**
  + /principals/list to list all principals records.
  + /principals/get/{id} to retrieve a specific principals record.
  + /principals/add to add a new principals record.
  + /principals/edit/{id} to update an existing principals record.
  + /principals/delete/{id} to delete a principals record.

**Entity: Principals**

* **Namespace:** App\Entity
* **Fields:**
  + id: Unique identifier for the principals record.
  + name: The name of the principal.
  + email: The email of the principal.
  + number: The contact number of the principal.

#### QualificationsController and Qualifications Entity

**Controller: QualificationsController**

* **Namespace:** App\Controller
* **Dependencies:**
  + EntityManagerInterface for managing database operations.
  + QualificationsRepository for interacting with the qualifications records.
* **Endpoints:**
  + /qualifications/list to list all qualifications records.
  + /qualifications/get/{id} to retrieve a specific qualifications record.
  + /qualifications/add to add a new qualifications record.
  + /qualifications/edit/{id} to update an existing qualifications record.
  + /qualifications/delete/{id} to delete a qualifications record.

**Entity: Qualifications**

* **Namespace:** App\Entity
* **Fields:**
  + id: Unique identifier for the qualifications record.
  + name: The name of the qualification.
  + description: A brief description of the qualification.

#### ScheduleController and Schedule Entity

**Controller: ScheduleController**

* **Namespace:** App\Controller
* **Dependencies:**
  + EntityManagerInterface for managing database operations.
  + ScheduleRepository for interacting with the schedule records.
* **Endpoints:**
  + /schedule/list to list all schedule records.
  + /schedule/get/{id} to retrieve a specific schedule record.
  + /schedule/add to add a new schedule record.
  + /schedule/edit/{id} to update an existing schedule record.
  + /schedule/delete/{id} to delete a schedule record.

**Entity: Schedule**

* **Namespace:** App\Entity
* **Fields:**
  + id: Unique identifier for the schedule record.
  + school\_id: The ID of the school.
  + student\_id: The ID of the student (nullable).
  + teacher\_id: The ID of the teacher (nullable).
  + monday: Schedule for Monday.
  + tuesday: Schedule for Tuesday.
  + wednesday: Schedule for Wednesday.
  + thursday: Schedule for Thursday.
  + friday: Schedule for Friday.

#### SchoolsController and Schools Entity

**Controller: SchoolsController**

* **Namespace:** App\Controller
* **Dependencies:**
  + EntityManagerInterface for managing database operations.
  + SchoolsRepository for interacting with the schools records.
* **Endpoints:**
  + /schools/list to list all schools records.
  + /schools/get/{id} to retrieve a specific schools record.
  + /schools/add to add a new schools record.
  + /schools/edit/{id} to update an existing schools record.
  + /schools/delete/{id} to delete a schools record.

**Entity: Schools**

* **Namespace:** App\Entity
* **Fields:**
  + id: Unique identifier for the schools record.
  + name: The name of the school.
  + address: The address of the school.
  + principal\_id: The ID of the principal.

#### StudentsController and Students Entity

**Controller: StudentsController**

* **Namespace:** App\Controller
* **Dependencies:**
  + EntityManagerInterface for managing database operations.
  + StudentsRepository for interacting with the students records.
* **Endpoints:**
  + /students/list to list all students records.
  + /students/get/{id} to retrieve a specific students record.
  + /students/add to add a new students record.
  + /students/edit/{id} to update an existing students record.
  + /students/delete/{id} to delete a students record.

**Entity: Students**

* **Namespace:** App\Entity
* **Fields:**
  + id: Unique identifier for the students record.
  + name: The name of the student.
  + email: The email of the student.
  + school\_id: The ID of the school.

#### SubjectsController and Subjects Entity

**Controller: SubjectsController**

* **Namespace:** App\Controller
* **Dependencies:**
  + EntityManagerInterface for managing database operations.
  + SubjectsRepository for interacting with the subjects records.
* **Endpoints:**
  + /subjects/list to list all subjects records.
  + /subjects/get/{id} to retrieve a specific subjects record.
  + /subjects/add to add a new subjects record.
  + /subjects/edit/{id} to update an existing subjects record.
  + /subjects/delete/{id} to delete a subjects record.

**Entity: Subjects**

* **Namespace:** App\Entity
* **Fields:**
  + id: Unique identifier for the subjects record.
  + name: The name of the subject.
  + description: A brief description of the subject.
  + school\_id: The ID of the school.

#### TeachersController and Teachers Entity

**Controller: TeachersController**

* **Namespace:** App\Controller
* **Dependencies:**
  + EntityManagerInterface for managing database operations.
  + TeachersRepository for interacting with the teachers records.
* **Endpoints:**
  + /teachers/list to list all teachers records.
  + /teachers/get/{id} to retrieve a specific teachers record.
  + /teachers/add to add a new teachers record.
  + /teachers/edit/{id} to update an existing teachers record.
  + /teachers/delete/{id} to delete a teachers record.

**Entity: Teachers**

* **Namespace:** App\Entity
* **Fields:**
  + id: Unique identifier for the teachers record.
  + name: The name of the teacher.
  + email: The email of the teacher.
  + school\_id: The ID of the school.

#### UsersController and Users Entity

**Controller: UsersController**

* **Namespace:** App\Controller
* **Dependencies:**
  + EntityManagerInterface for managing database operations.
  + UsersRepository for interacting with the users records.
* **Endpoints:**
  + /users/list to list all users records.
  + /users/get/{id} to retrieve a specific users record.
  + /users/add to add a new users record.
  + /users/edit/{id} to update an existing users record.
  + /users/delete/{id} to delete a users record.

**Entity: Users**

* **Namespace:** App\Entity
* **Fields:**
  + id: Unique identifier for the users record.
  + username: The username of the user.
  + email: The email of the user.
  + password: The password of the user.

This documentation provides an overview of how each controller interacts with its corresponding entity to perform CRUD operations. Each controller uses the EntityManagerInterface for managing database operations and a repository class for interacting with the entity records. The endpoints in the controllers provide RESTful API methods for listing, retrieving, adding, editing, and deleting records.

### Database Interaction

## Basic Queries

**[sql]**

SELECT \* FROM schedules;

## Inserting a New Mark

To insert a new mark into the marks table , you can use the following MySQL query as an example:

**[sql]**

INSERT INTO marks (mark\_id, mark, date\_assessed, students\_student\_id, subjects\_subject\_id) VALUES (2, '78', '2024-06-01', 2, 2);

## More Complex Queries

Average Marks of Each Student by Subject

**[sql]**

SELECT

s.student\_id,

s.name AS student\_name,

sub.subject\_name,

AVG(m.mark) AS average\_mark

FROM students s

JOIN marks m ON s.student\_id = m.students\_student\_id

JOIN subjects sub ON m.subjects\_subject\_id = sub.subject\_id

GROUP BY

s.student\_id, sub.subject\_name

ORDER BY

s.student\_id, sub.subject\_name;

List All Teachers with Their Qualifications, Schools They Teach At, and the Subjects They Teach

**[sql]**

SELECT

t.teacher\_id,

t.name AS teacher\_name,

t.email AS teacher\_email,

q.qualification\_name,

sc.name AS school\_name,

GROUP\_CONCAT(sub.subject\_name SEPARATOR ', ') AS subjects\_taught

FROM

teachers t

JOIN

qualifications q ON t.qualifications\_qualification\_id = q.qualification\_id

JOIN

schools sc ON t.schools\_school\_id = sc.school\_id

JOIN

teachers\_subjects ts ON t.teacher\_id = ts.teacher\_id

JOIN

subjects sub ON ts.subject\_id = sub.subject\_id

GROUP BY

t.teacher\_id, t.name, q.qualification\_name, sc.name

ORDER BY

t.teacher\_id;

### Testing

Each controller in the system (AttendanceController, GradesController, MarksController, ParentsController, PrincipalsController, QualificationsController, ScheduleController, SchoolsController, StudentsController, SubjectsController, TeachersController, UsersController) handles similar operations related to managing different entities. The following detailed descriptions cover the common tests performed across these controllers:

#### Test: Adding Record

**Description:**  
This test validates the functionality of adding a new record to the system.

**Steps:**

1. **Send Request:** A POST request is sent to the add endpoint (e.g., /entity/add) with the necessary data for the entity being tested. The data is sent in JSON format and includes relevant fields such as name, email, date, status, etc.
2. **Check Response Status:** The test verifies that the response status code is 201 (Created), indicating that the record was successfully added.
3. **Validate Response Format:** The response content is checked to ensure it is in JSON format.
4. **Check Success Message:** The response is checked for a success message indicating that the record was created successfully.

**Example:** Adding a new mark record:

* Endpoint: /marks/add
* Data: { "mark": 85, "date": "2024-06-15", "student\_id": 1, "subject\_id": 1 }
* Expected Status: 201 (Created)
* Expected Message: "Mark created successfully"

#### Test: Listing Records

**Description:**  
This test validates the functionality of listing all records for a specific entity.

**Steps:**

1. **Send Request:** A GET request is sent to the list endpoint (e.g., /entity/list).
2. **Check Response Status:** The test verifies that the response status code is 200 (OK), indicating that the request was successful.
3. **Validate Response Format:** The response content is checked to ensure it is in JSON format.
4. **Validate Content:** Additional assertions can be added to ensure the response contains the expected records.

**Example:** Listing all marks:

* Endpoint: /marks/list
* Expected Status: 200 (OK)

#### Test: Getting Specific Record

**Description:**  
This test validates the functionality of retrieving a specific record by its ID.

**Steps:**

1. **Send Request:** A GET request is sent to the get endpoint with the record ID as a parameter (e.g., /entity/get/{id}).
2. **Check Response Status:** The test verifies that the response status code is 200 (OK), indicating that the record was successfully retrieved.
3. **Validate Response Format:** The response content is checked to ensure it is in JSON format.
4. **Validate Content:** Additional assertions can be added to ensure the response contains the correct data for the specified ID.

**Example:** Getting a specific mark by ID:

* Endpoint: /marks/get/1
* Expected Status: 200 (OK)

#### Test: Editing Record

**Description:**  
This test validates the functionality of editing an existing record in the system.

**Steps:**

1. **Send Request:** A POST request is sent to the edit endpoint with the record ID as a parameter and the updated data in JSON format (e.g., /entity/edit/{id}).
2. **Check Response Status:** The test verifies that the response status code is 200 (OK), indicating that the record was successfully updated.
3. **Validate Response Format:** The response content is checked to ensure it is in JSON format.
4. **Check Success Message:** The response is checked for a success message indicating that the record was updated successfully.

**Example:** Editing a mark record:

* Endpoint: /marks/edit/1
* Data: { "mark": 90, "date": "2024-06-20", "student\_id": 1, "subject\_id": 1 }
* Expected Status: 200 (OK)
* Expected Message: "Mark updated successfully"

#### Test: Deleting Record

**Description:**  
This test validates the functionality of deleting a record from the system.

**Steps:**

1. **Send Request:** A DELETE request is sent to the delete endpoint with the record ID as a parameter (e.g., /entity/delete/{id}).
2. **Check Response Status:** The test verifies that the response status code is 200 (OK), indicating that the record was successfully deleted.
3. **Validate Response Format:** The response content is checked to ensure it is in JSON format.
4. **Check Success Message:** The response is checked for a success message indicating that the record was deleted successfully.

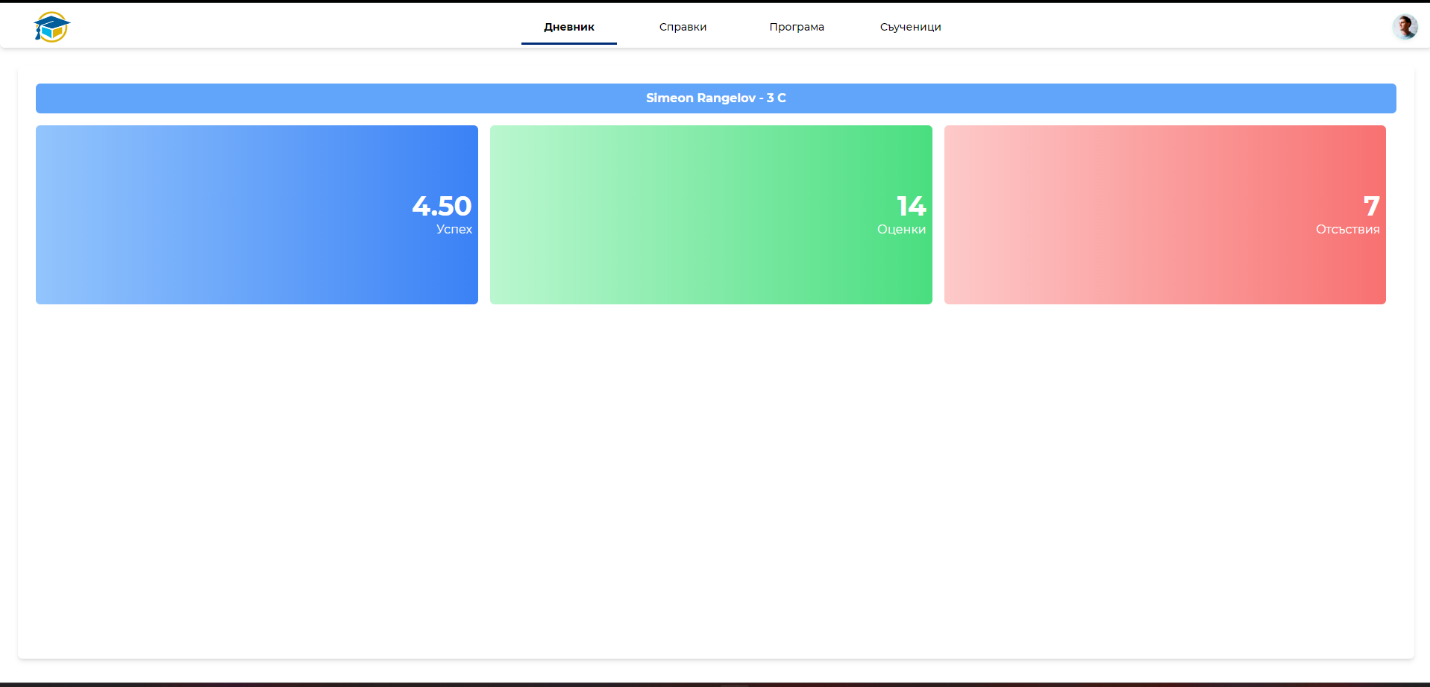
**Example:** Deleting a mark record:

* Endpoint: /marks/delete/1
* Expected Status: 200 (OK)
* Expected Message: "Mark deleted successfully"

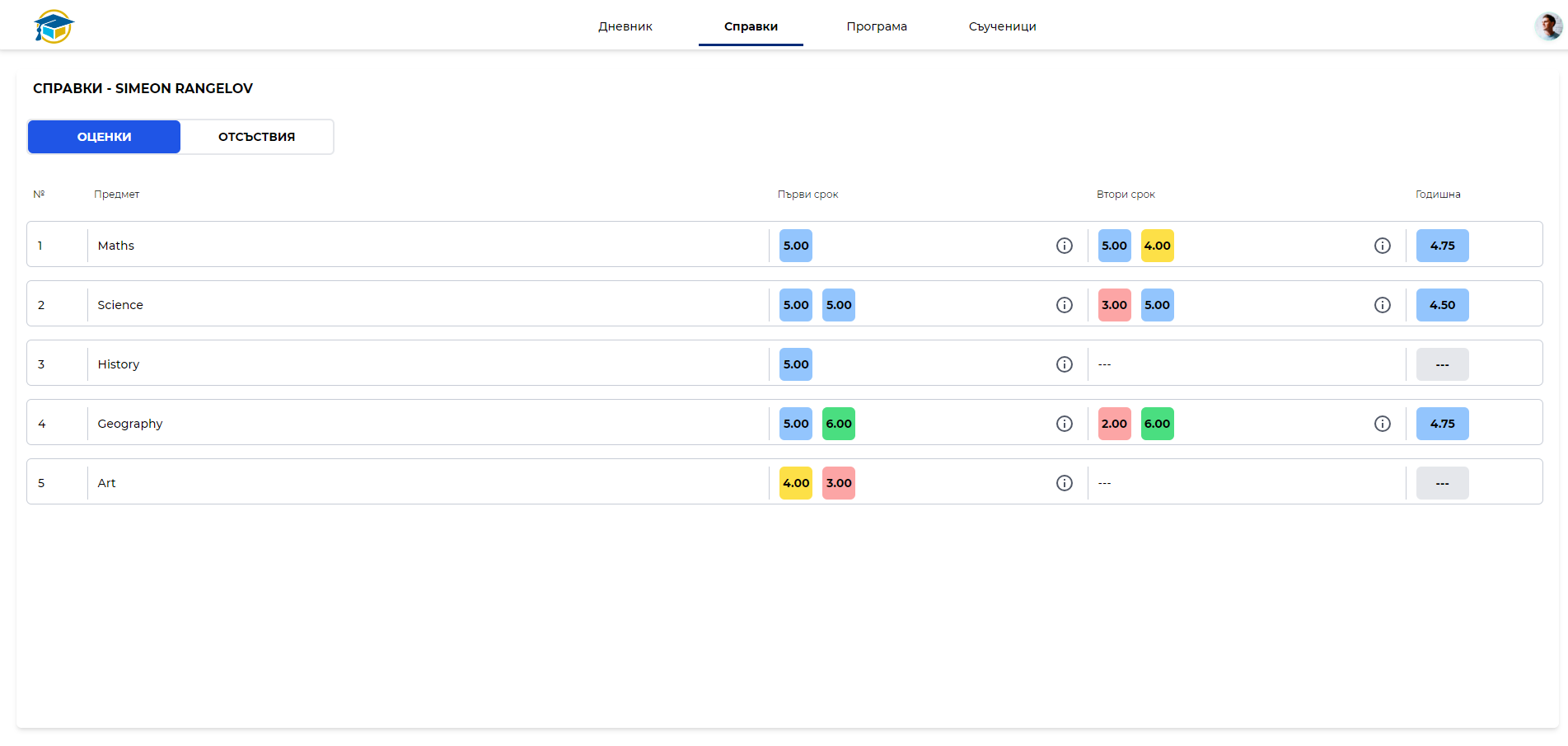
### User Manual

#### Student

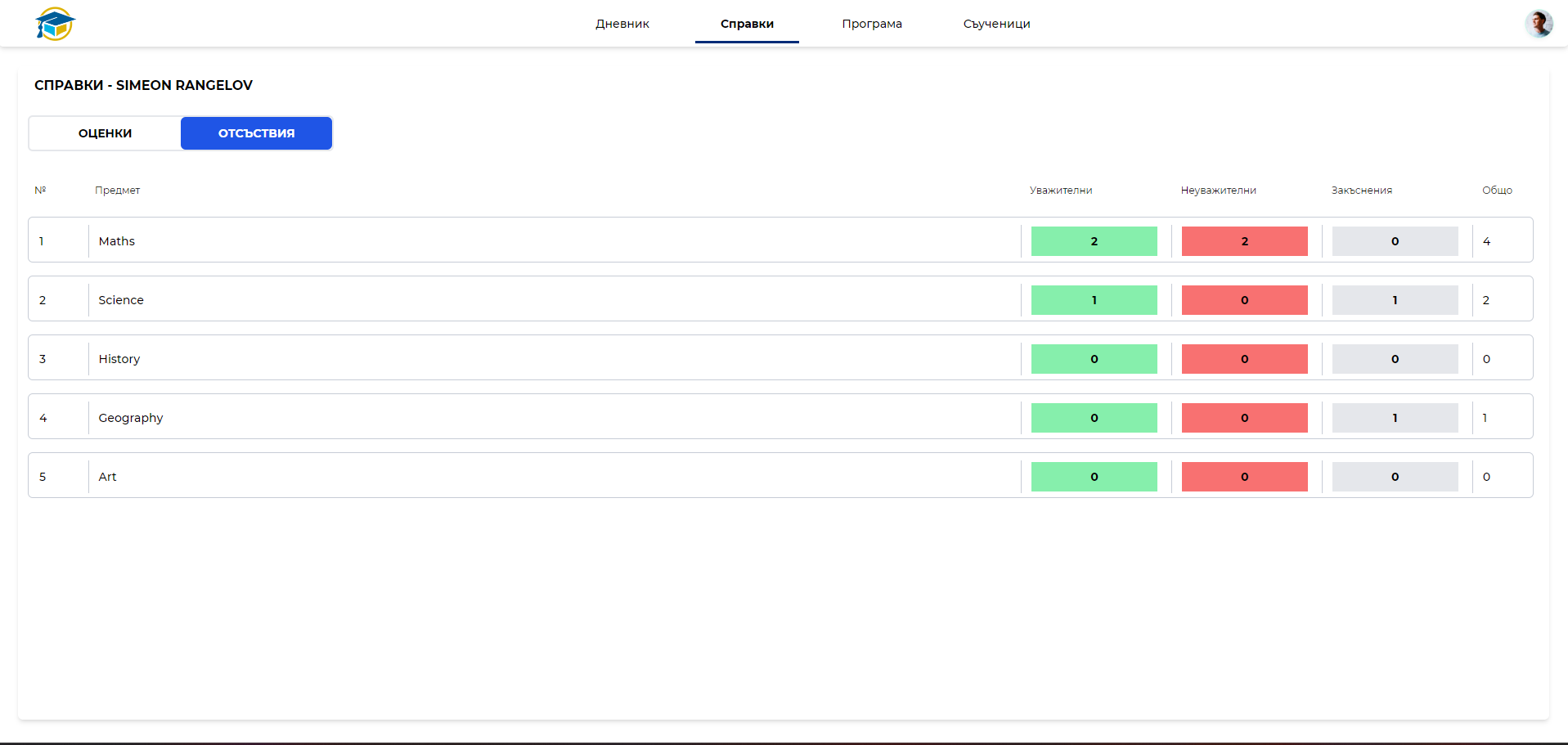
After successfully logging in/registering as a **student** , this would be your main dashboard:



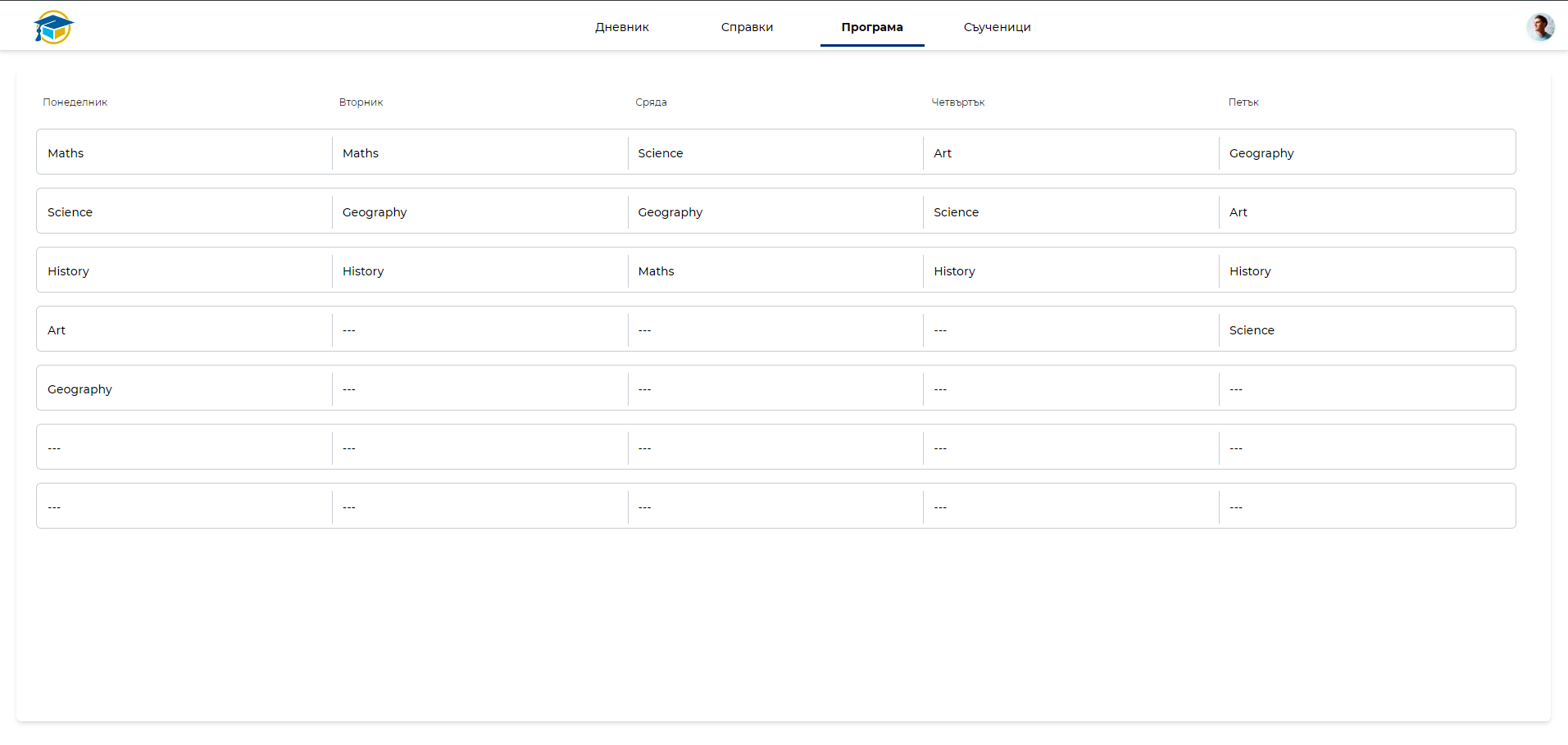
Справки:



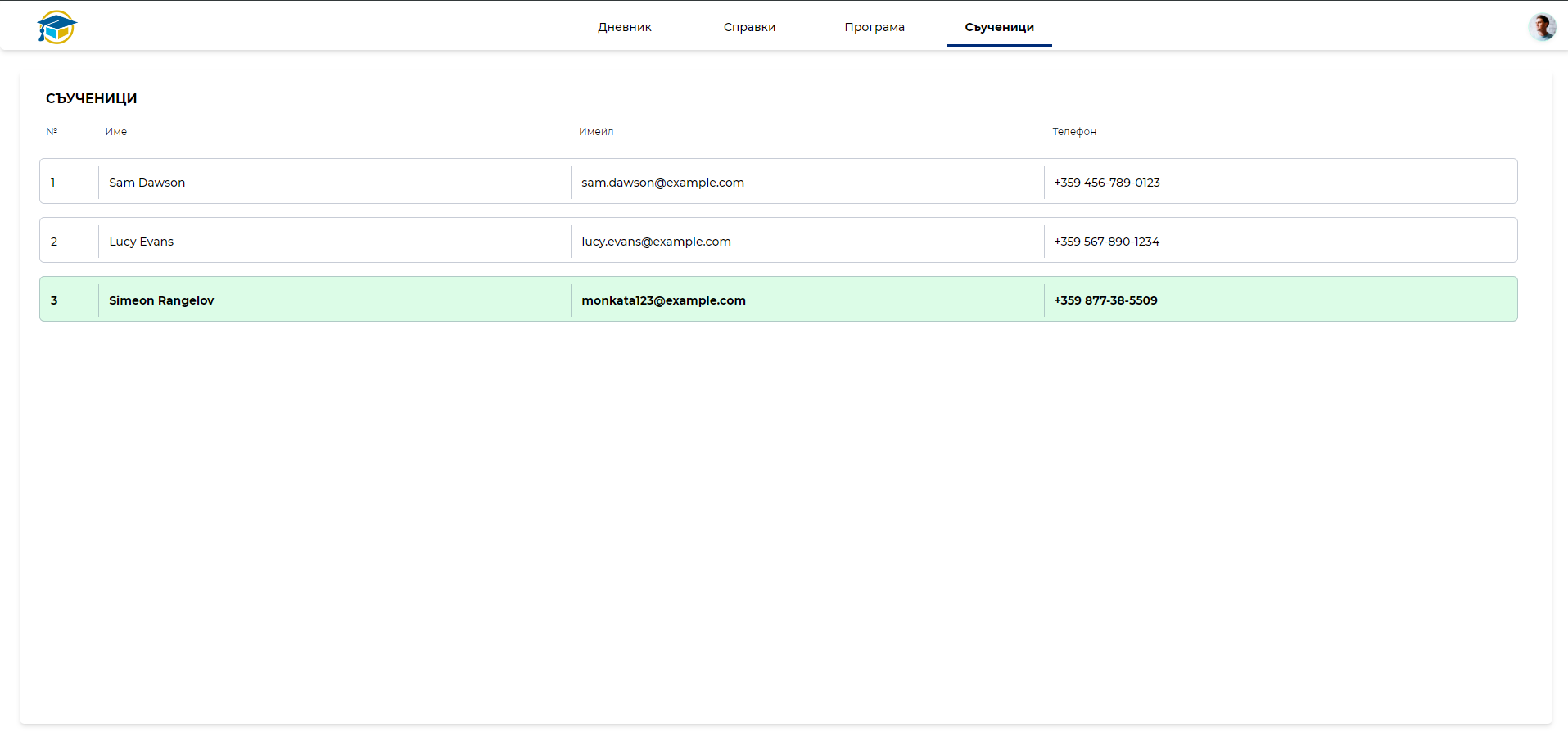
Справки-отсъствия



Програма:

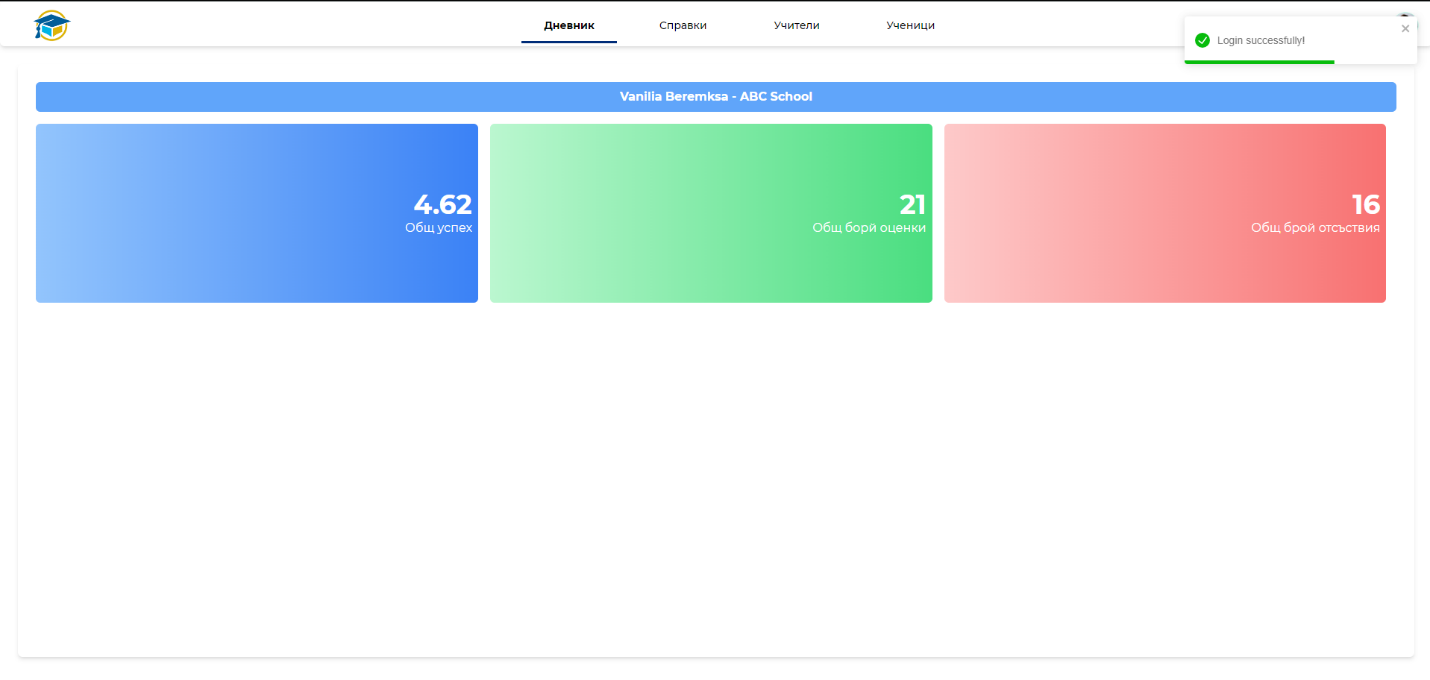


Съученици

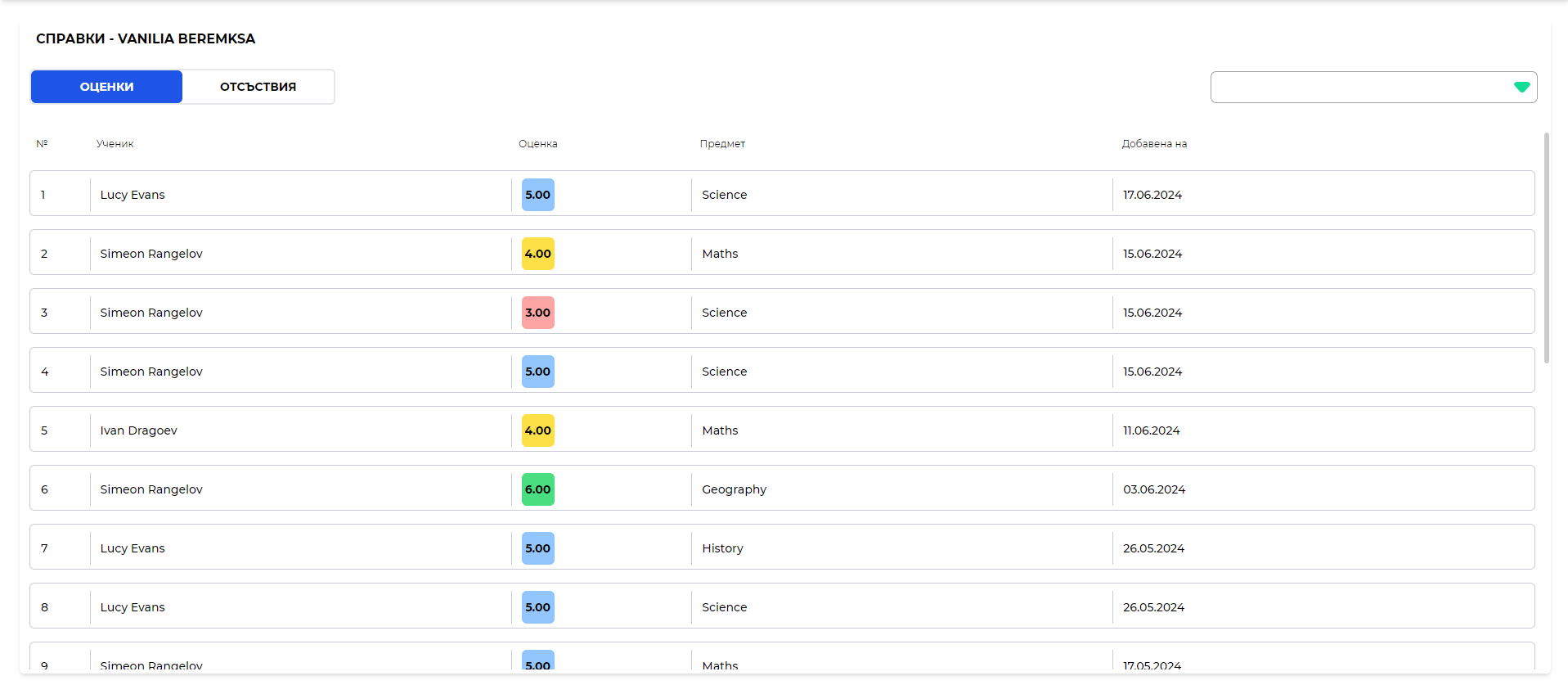


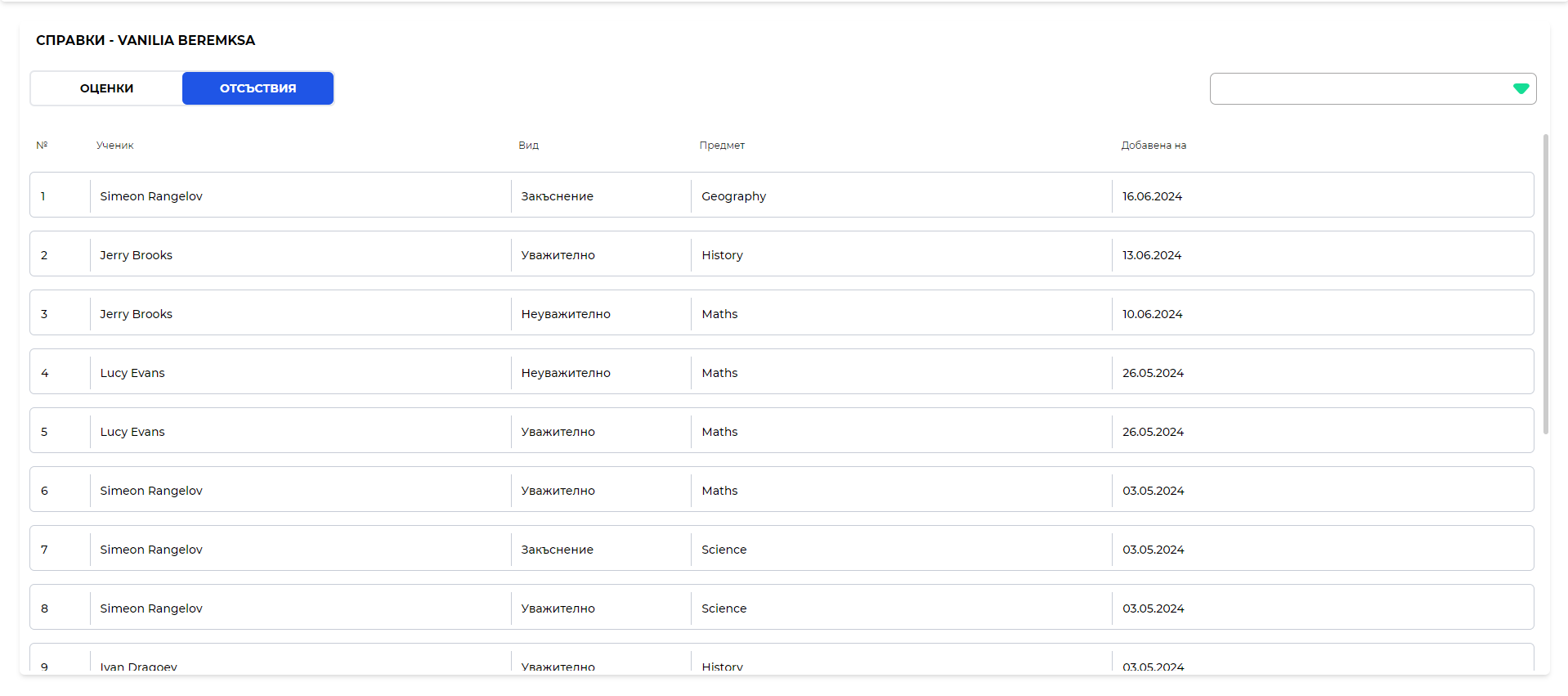
#### Principal

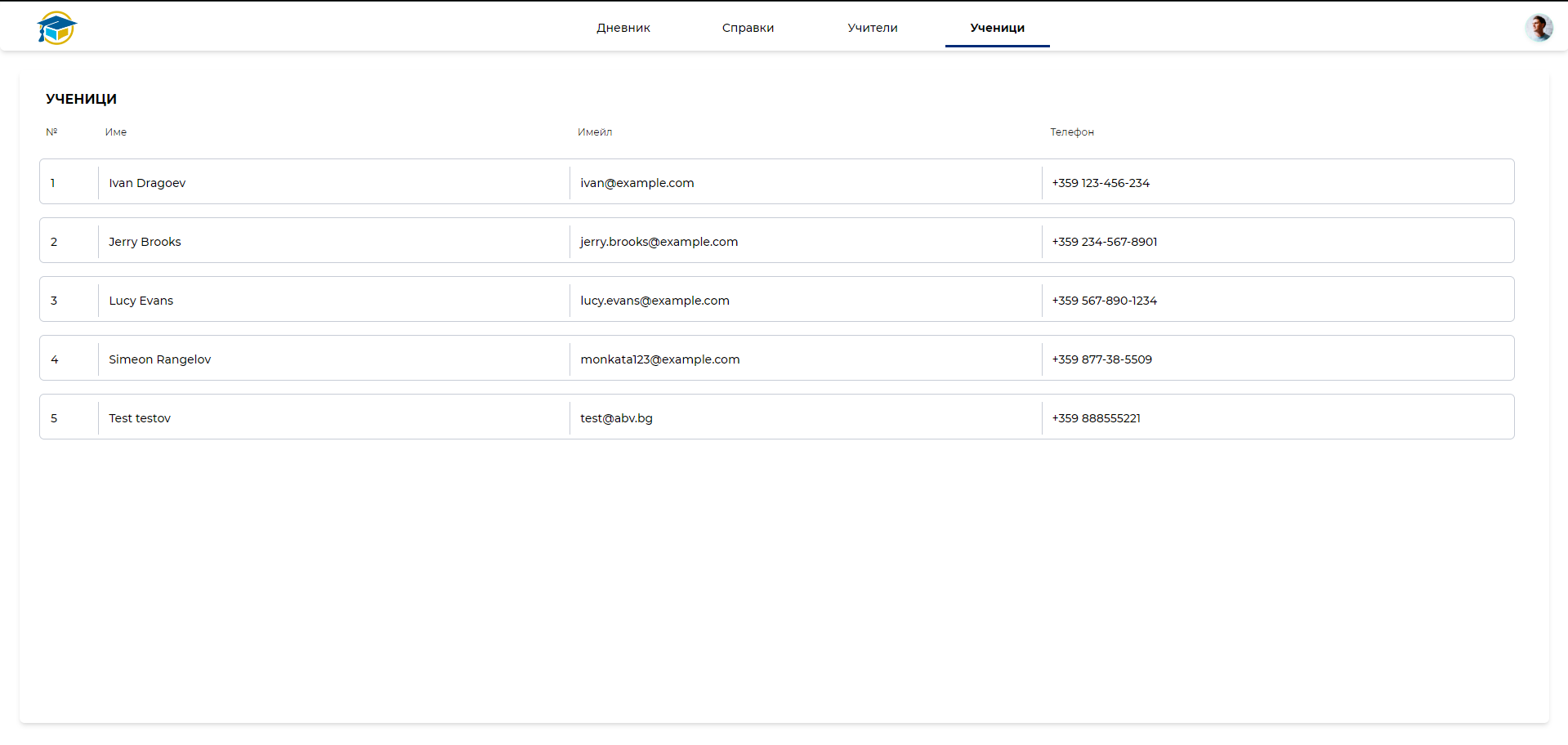
DashBoard displaying information for a specific school



Справки - оценки

  
Справки – отсъствия



Ученици 

Учители

