3-7-2023

AuleWeb

Web Engineering

JUAN JOSE GOMEZ BORRALLO
FRANCISCO DE ASIS BERMUDEZ CAMPUZANO
WEB ENGINEERING

Content

Software dependencies

The main dependencies that we have used in the server are:

1. Mssql-jdbc-12.2.0.jre11 & mysql-connector-j-8.0.33. Both dependencies are used to successfully connect to the database.
2. Commons-fileupload-1.5. This dependency is used to process and handle uploaded files in web applications
3. Fullcalendar: The fullcalendar library has been used to generate the skeleton of the calendar. And through calls in the backend and JS, the calendar has been configured to be functional. Link: https://fullcalendar.io/
4. The rest of the dependencies are native to Apache Netbeans. They already come when the version recommended by the project requirements is installed.

All dependencies can be downloaded from https://mvnrepository.com/

Functions

We are going to name the functions used in the project according to the structure that we have built.

SOURCE PACKAGES:

com.mycompany.aulaweb: Within this package we find all the Servlets that have been used to carry out the project. These Servlets are responsible for collecting the data from the frontend and passing it to the database and vice versa. Each Servlet uses some function created for reading or writing the database. We will define these functions later.

sql: In this package we can find all the classes of the database and the functions that we use to be able to read, insert, update and delete data from the web. We can find the following classes:

1. Admin.java:
   1.1. Admin(): It is the constructor of the Admin class, to which it is passed as parameters (id, email, password)
   1.2. getEmail(): This function returns the email associated.
   1.3. getPassword(); This function returns the password associated.
   1.4. toString(): This function overrides the default toString() method of the class and returns a JSON-formatted string representation of the object.
2. Aula.java:
   2.1. Aula(int id, String nombre, int aforo): This constructor initializes an object of the Aula class with the provided id, nombre, and aforo values.
   2.2. Aula(int id, String nombre, String descripcion, String ubicacion, int aforo, int numEnchufes, int red, Boolean tieneProyector, Boolean tienePantallaMotorizada, Boolean tienePantallaManual, Boolean tieneSisAudio, Boolean tienePC, Boolean tieneMicIna, Boolean tieneMicAla, Boolean tieneRetroProy, Boolean tieneWifi): This constructor initializes an object of the Aula class with the provided values for various properties, including id, nombre, descripcion, ubicacion, aforo, numEnchufes, red, and several boolean flags for different features.

    2.3. toString(): This function overrides the default toString() method of the class and returns a JSON-formatted string representation of the object. It includes the object's id, nombre, and aforo as properties in the JSON string.
    2.4. toStringComplete(): This function returns a detailed JSON-formatted string representation of the object, including additional properties such as descripcion, ubicacion, numEnchufes, red, and various boolean flags for features.
3. Deparment.java
    3.1. Deparment(int id, String nombre): This constructor initializes an object of the Deparment class with the provided id and nombre values.
    3.2. toString(): This function overrides the default toString() method of the class and returns a JSON-formatted string representation of the object. It includes the object's id and nombre as properties in the JSON string.
4. Evento.java
    4.1. Evento(int id, String nombre, String descripcion, String tipo, String fechaInicio, String fechaFin): This constructor initializes an object of the Evento class with the provided id, nombre, descripcion, tipo, fechaInicio, and fechaFin values.
    4.2. Evento(int id, String nombre, String descripcion, String nombreResponsable, String emailResponsable, String tipo, String fechaInicio, String fechaFin, String recurrencia, String fechaFinRecurrencia): This constructor initializes an object of the Evento class with the provided id, nombre, descripcion, nombreResponsable, emailResponsable, tipo, fechaInicio, fechaFin, recurrencia, and fechaFinRecurrencia values.
    4.3. toString(): This function overrides the default toString() method of the class and returns a JSON-formatted string representation of the object. It includes the object's id, nombre, descripcion, tipo, fechaInicio, and fechaFin as properties in the JSON string. The fechaInicio and fechaFin values are formatted to replace the space with "T" to adhere to the standard format.
    4.4. toStringComplete(): This function returns a JSON-formatted string representation of the object, including additional properties like nombreResponsable, emailResponsable, recurrencia, and fechaFinRecurrencia. The fechaInicio and fechaFin values remain unmodified in this method.
5. SQLConstructor: Contains all the functions that make calls to the database.
    5.1. readEvents(int idAula, String nombre, String descripcion, String nombreResponsable, String emailResponsable, String fechaInicio, String fechaFin, String tipo, String recurrencia, String fechaFinRecurrencia): This method inserts a new event into the database by constructing an SQL INSERT statement and executing it.
    5.2. editEvents(int idEvento, String nombre, String descripcion, String nombreResponsable, String emailResponsable, String fechaInicio, String fechaFin, String tipo, String recurrencia, String fechaFinRecurrencia): This method updates an existing event in the database by constructing an SQL UPDATE statement and executing it.
    5.3. insertSql(String sql): This method executes an SQL statement provided as a parameter. It establishes a connection to the database, creates a statement object, and executes the SQL statement.
    5.4. exeQuery(): This method executes a SELECT query on the "eventos" table and prints the retrieved data.
    5.5. checkUserExistence(String param1, String param2): This method checks the existence of a user in the "login" table based on two parameters. It constructs a parameterized SQL SELECT statement and retrieves the count of matching rows.

5.6. deleteEvento(int id): This method deletes an event from the "eventos" table based on the specified event ID.
5.7. exeQueryDepartamentos(): This method executes a SELECT query on the "edificios" table and retrieves the data. It constructs an ArrayList of department objects and returns the result as a formatted string.
5.8. exeQueryAdmins(): This method executes a SELECT query on the "login" table and retrieves the data. It constructs an ArrayList of admin objects and returns the result as a formatted string.
5.9. CredentialChecker(): This method takes a string parameter and returns a boolean value.
5.10. exeQueryAulasId(int id): This method executes a SELECT query on the "aulas" table with a specified ID parameter. It constructs an ArrayList of aula objects and returns the result as a formatted string.
5.11. exeQueryEventos(int id): This method executes a SELECT query on the "eventos" table with a specified ID parameter. It constructs an ArrayList of evento objects and returns the result as a formatted string.
5.12. getEventId(int id): This method retrieves the details of an event from the "eventos" table based on the specified event ID. It constructs an evento object and returns the result as a formatted string.
5.13. FormatFecha(Date fecha): This private helper method formats a given date object to a string in the format "yyyy-MM-dd".
5.14. generateEventsCSV2(Date fechaInicio, Date fechaFin, String filePath): This method generates a CSV file containing events within a specified date range. It executes a parameterized SELECT query, retrieves the data, and writes it to a CSV file.
5.15. generateAulas(String filePath): This method generates a CSV file containing aulas data. It executes a SELECT query on the "aulas" table, retrieves the data, and writes it to a CSV file.
5.16. escapeCSVValue(String value): This private static method takes a value as input and escapes it for CSV format. If the value is null, an empty string is returned. If the value contains a comma, it is enclosed in double quotes before returning.
5.17. readDataBuildings(): This method reads data from a CSV file named "edificios.csv" and inserts it into the "edificios" table in the database. It uses FileReader and BufferedReader to read the file line by line. Each line is split using the semicolon (;) separator, and the data is used to construct an SQL INSERT statement. The statement is then executed using SQLConstructor.insertSql().
5.18. readDataAules2(): This method reads data from a CSV file named "aulas.csv" and inserts it into the "aulas" table in the database. Similar to readDataBuildings, it reads the file line by line, splits each line using the semicolon separator, and constructs an SQL INSERT statement using the data. The statement is executed using SQLConstructor.insertSql().
5.19. readDataAdmin(): This method reads data from a CSV file named "admins.csv" and inserts it into the "login" table in the database. It follows a similar approach as the previous methods, reading the file line by line, splitting each line using the semicolon separator, and constructing an SQL INSERT statement with the data. The statement is executed using SQLConstructor.insertSql().
5.20. readDataAules(File file): This method takes a File object as a parameter, representing the CSV file to be read. It uses a FileReader and BufferedReader to read

the file line by line. The first line is skipped as it typically contains column headers. For each subsequent line, the data is split using the semicolon (;) separator. It then checks if the record already exists in the database by calling the registroExiste method. If the record exists, it calls the actualizarRegistro method; otherwise, it calls the insertarRegistro method. After processing each line, it removes the processed record from the registrosActuales list. Finally, it calls the eliminarRegistrosNoExistentes method to delete any remaining records in the database that do not exist in the CSV file.

5.21. readDataDepartments(File File): This function reads data from a CSV file and processes it to update or insert records in a database.

5.22. obtenerRegistrosActualesDepartment(): This function connects to the database and queries all the current records from the "edificios" table. It returns a list of the existing record IDs.

5.23. eliminarRegistrosNoExistentesDepartment(List<String> registrosNoExistentes): This function receives a list of records that do not exist in the file and deletes them from the corresponding "edificios" tables in the database. It uses prepared statements to efficiently delete the records.

5.24. insertarRegistroDepartment(String[] datos): This function is responsible for inserting a new record into the "edificios" table in the database. It takes an array of data as input, where datos[0] represents the ID and datos[1] represents the nombre (name) of the record. The function constructs an SQL INSERT statement with the provided data and executes it using SQLConstructor.insertSql(sql).

5.25. actualizarRegistroDepartments(String[] datos): This function updates an existing record in the "edificios" table in the database. It also takes an array of data as input, where datos[0] represents the ID and datos[1] represents the new nombre (name) for the record. The function constructs an SQL UPDATE statement with the provided data and executes it using SQLConstructor.insertSql(sql). The update statement sets the new ID and nombre for the record identified by the existing ID (datos[0]).

5.26. obtenerRegistrosActuales(): This private static method connects to the database and retrieves a list of the current records' IDs from the aulas table. It uses a Connection, PreparedStatement, and ResultSet to execute an SQL SELECT query and retrieves the IDs. The IDs are added to a List<String> and returned.

5.27. eliminarRegistrosNoExistentes(List<String> registrosNoExistentes): This private static method connects to the database and deletes records from the eventos and aulas tables that do not exist in the registrosNoExistentes list. It uses separate PreparedStatement objects to execute DELETE queries for each table, iterating over the registrosNoExistentes list and binding the ID parameter accordingly.

5.28. registroExiste(String idAula): This private static method connects to the database and checks if a record with the given idAula exists in the aulas table. It executes an SQL SELECT query with a WHERE clause to count the number of matching records. If the count is greater than 0, it returns true, indicating that the record exists; otherwise, it returns false.

5.29. insertarRegistro(String[] datos): This method takes an array of datos as a parameter, which contains the data for a new record to be inserted. The method constructs an SQL INSERT statement using string concatenation to include the values from the datos array. The SQL statement inserts the values into the corresponding columns of the aulas table. The constructed SQL statement is then printed, and the insertSql method from the SQLConstructor class is called to execute the SQL statement.

5.30. actualizarRegistro(String[] datos): This method is similar to insertarRegistro, but it constructs an SQL UPDATE statement to update an existing record in the aulas table. The SQL statement includes the values from the datos array, updating the corresponding columns based on the provided record ID (datos[0]). The constructed SQL statement is printed, and the insertSql method from the SQLConstructor class is called to execute the SQL statement.

We will add a quick refresher on functions in js. These are distributed throughout the different html files, inside the additional_pages folder:

loadEventos(callBack):

- It makes a fetch request to a specific URL, passing the id as a query parameter.
- The response is handled as JSON and passed to the callback function (paintEvents).

paintEvents(data):

- It initializes variables rows and csvContent for storing HTML rows and CSV data, respectively.
- It iterates over each evento in the data array and creates HTML table rows and CSV rows based on the event's properties.
- The generated rows are appended to rows and csvContent, respectively.
- The HTML table body is updated with the generated rows.
- A download link for the CSV file is created and appended to the document body.
- The first event's field values are assigned to individual variables.
- The editUrl is constructed using the stored field values and the event's ID.
- The "Edit Event" button's click event is updated to redirect to the editUrl when clicked.
- There is also a goToNuevoEvento function that redirects to a "new event" page, passing the idAula and idDepartamento as query parameters.

The **loadAulas** function makes a fetch request to a server endpoint (/AuleWeb/AulasServlet?id=${id}) to retrieve a list of rooms (aulas). The response is converted to JSON, and the loadSelectAulas function is called with the resulting data.

The **loadEventos** function makes a fetch request to a server endpoint (/AuleWeb/EventosServlet?id=${idAula}) to retrieve a list of events associated with a specific room (aula). The response is converted to JSON, and the createCalendario function is called with the resulting data.

The **loadSelectAulas** function takes an array of rooms (aulas) as a parameter and populates the select element with the ID "select_aulas" with options based on the room data. It also sets the selected option based on the idAulaParam or the first room in the array. It then calls the loadEventos function with the selected option's ID and the createCalendario function.

The **paintDetalleAula** function updates the HTML elements with details about the selected room (aula) based on the selected option's attributes.

The **MapToFullCalendarEvent** function takes an array of events and maps them to FullCalendar event objects with specific properties like ID, title, start date, end date, and color based on the event type.

The **goToNuevoEvento** function redirects the user to a new event page with the selected department ID (idDepartamento) and the ID of the selected room (idAula) as query parameters.

The **createCalendario** function initializes a FullCalendar instance and configures its options. It sets up the calendar to display a weekly view (timeGridWeek), defines the time slot duration, minimum and maximum times, and whether to show an all-day slot. It also sets the events using the MapToFullCalendarEvent function and handles event clicks by redirecting the user to a test page with the selected department ID and the clicked event's ID as query parameters.

Additional features

In the project we have added the following extra functionality:

1. Once you are viewing the calendar, you can click on the desired event, and you will be redirected to another tab where you can see all the properties of said event, and also be able to download the specific data of that event in CSV format.
2. When you are going to edit an event, the data of the event you have selected will appear already written, so it will be easier for you to know what data you are processing and what you want to modify, without having to memorize the data that we had previously entered at the time to create the event.
3. We have added some validations in the backend related to the insertion of events to check that the times that are entered and the data format are correct.
4. We have added a validation in the backend related to the upload of classrooms. When you want to upload a new CSV document with the modified or added classrooms, a check is made to verify that the data and format are correct.
5. When you access the page where the calendar is located (when selecting a department), the properties of the classroom that is selected have been added just to the right of the calendar. The available classrooms can be seen by clicking on the dropdown in the upper area.

Relational scheme DB

The database consists of the following tables:

1. The "edificios" table represents the buildings where the classrooms are located. Each record in this table contains a unique identifier (id) and the name of the building.

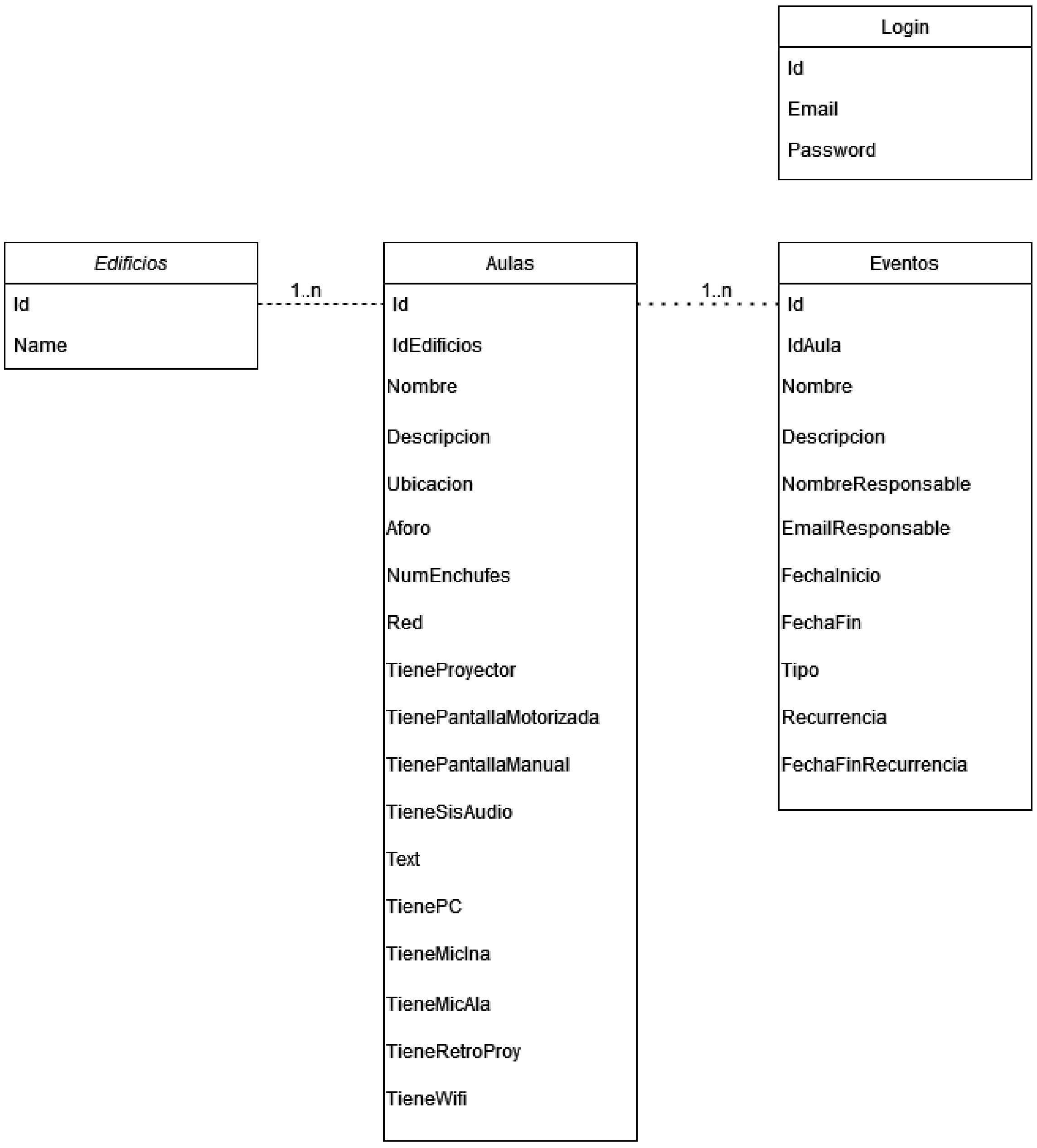
2. The "aulas" table contains information about the classrooms in the buildings. Each classroom is associated with a building through its building identifier (idEdificios). The fields of the table include a unique identifier (id), the name of the classroom, a description, the location within the building, the seating capacity, the number of available power outlets, the availability of network connections, and optional attributes such as the presence of a projector, motorized screens, manual screens, audio system, PC, wireless microphone, wired microphone, overhead projector, and Wi-Fi access.

3. The "eventos" table stores information about the events taking place in the classrooms. Each event is associated with a specific classroom through its classroom identifier (idAula). The fields of the table include a unique identifier (id), the name of the event, a detailed description, the name of the event organizer, their email address, the start and end dates of the event, the type of event (such as conference, exam, seminar, etc.), the recurrence of the event (daily, weekly, monthly, or none), and the end date of the recurrence.

4. The "login" table stores information about authorized users who have administrative privileges to access and work with events. This table specifically holds the email address, which serves as the unique identifier for administrators, and their corresponding login password. Administrators with valid login credentials can access and perform administrative functions on the events within the system.

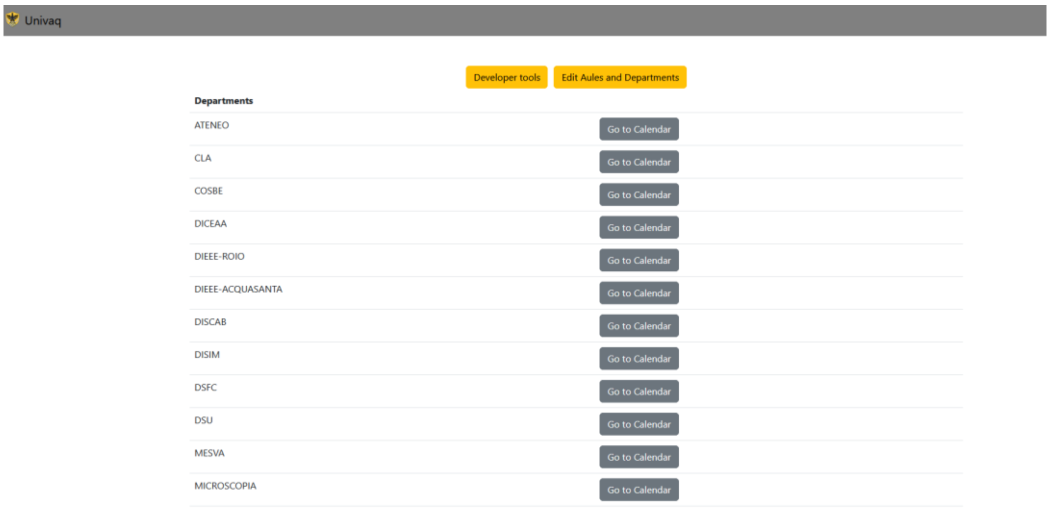
These tables represent the structure of the database and the relationships between them. The "edificios" and "aulas" tables are related in a "one-to-many" (1..n) relationship, meaning that a building can contain multiple classrooms, but each classroom belongs to a single building. Similarly, the "aulas" table and the "eventos" table also have a "one-to-many" (1..n) relationship, indicating that a classroom can host multiple events, but each event is associated with a single classroom.

This is the basic design of the database and its relationships, providing an organized structure for storing information about buildings, classrooms, and events.



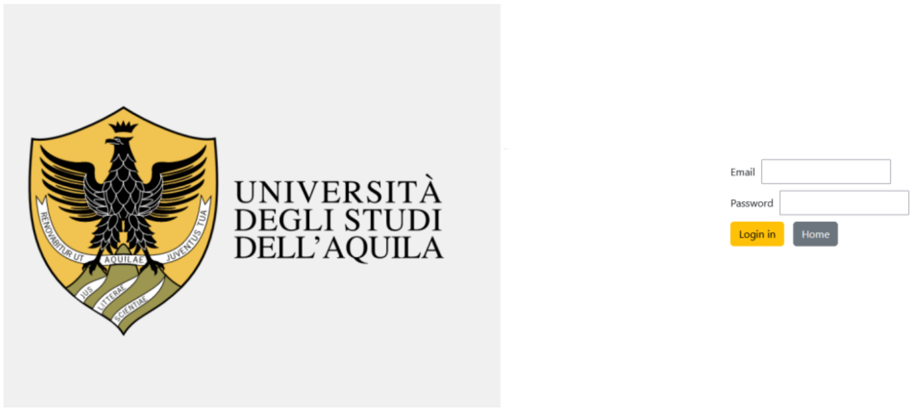
Design analytic description

Our company's application homepage utilizes dynamic data from a database to showcase departments. This approach ensures that the department information is always up-to-date and allows for easy scalability and maintenance of the application. By fetching department data dynamically, we can effortlessly add or modify departments without modifying the source code. This enhances the flexibility and efficiency of our application. (http://localhost:8080/AuleWeb/index.html)



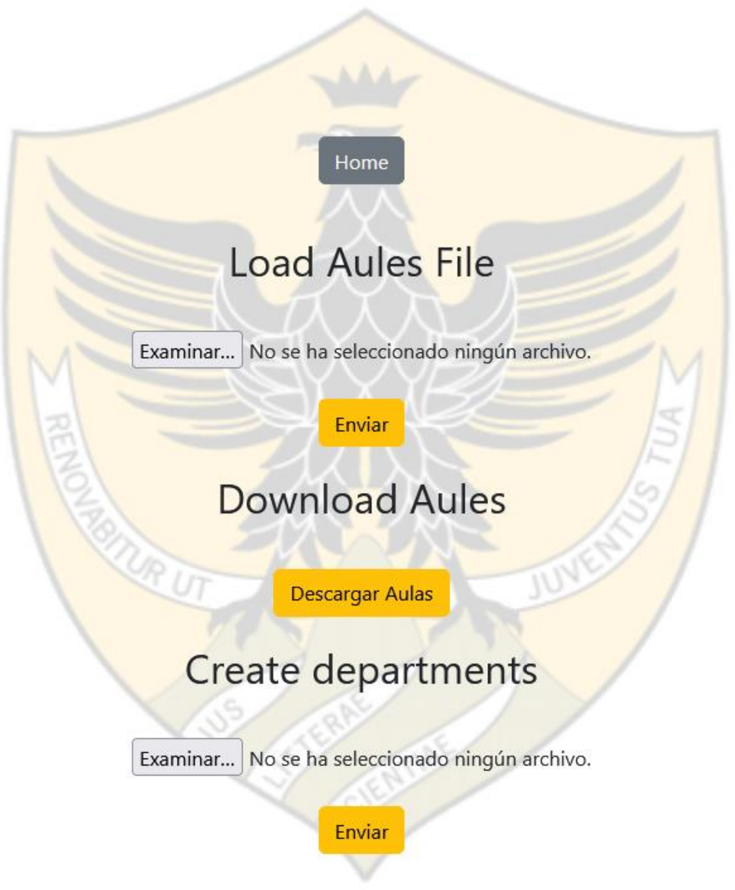
Departments	
ATENEO	Go to Calendar
CLA	Go to Calendar
COSBE	Go to Calendar
DICEAA	Go to Calendar
DIEEE-ROIO	Go to Calendar
DIEEE-ACQUASANTA	Go to Calendar
DISCAB	Go to Calendar
DISIM	Go to Calendar
DSFC	Go to Calendar
DSU	Go to Calendar
MESVA	Go to Calendar
MICROSCOPIA	Go to Calendar

Our login page accesses our database, checking the email and password, accessed by Developer Tools Button. (http://localhost:8080/AuleWeb/additional_pages/adminlogin.html)



We modify our data stored in the db without accessing the code, automatically through csv files.

(http://localhost:8080/AuleWeb/additional_pages/editAules.html)


Home
Load Aules File
Examinar... No se ha seleccionado ningún archivo.
Enviar
Download Aules
Descargar Aulas
Create departments
Examinar... No se ha seleccionado ningún archivo.
Enviar

Our application's calendar page features a dynamic calendar that retrieves event data from our database. This dynamic functionality allows us to load events in real-time, providing users with up-to-date scheduling information. We have implemented features that allow users to add, modify, and delete events directly from the application, ensuring seamless event management.

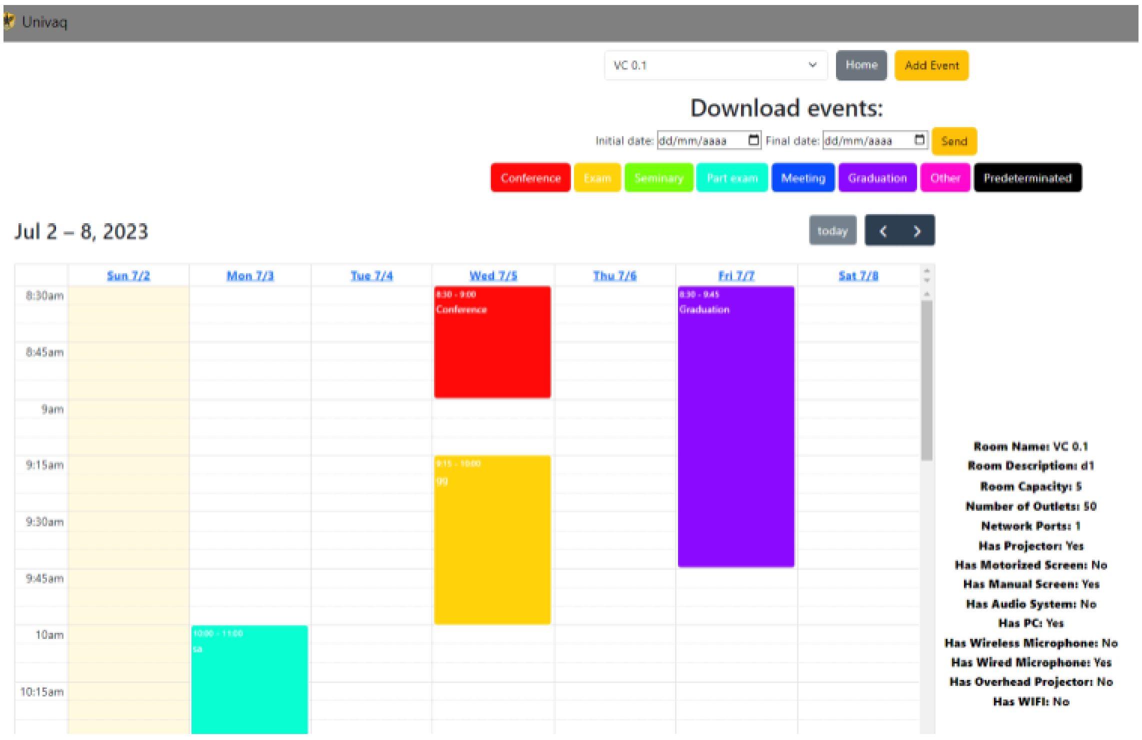
Additionally, the calendar page incorporates a color legend to signify the different event types. This legend remains static and provides a visual reference for users to easily identify and understand the meaning behind each color assigned to specific event types.

By combining static elements like the color legend with dynamic features such as event retrieval, modification, and deletion from the database, our calendar page delivers a comprehensive and interactive scheduling experience for users.

In addition, our calendar page offers the functionality to download events in CSV format within a specified time period. This feature allows users to generate a CSV file containing event data for a selected range of dates directly from the page. By providing this download capability, users can easily export event information and further analyze or share it as needed.

This dynamic feature complements the overall functionality of the calendar page, enhancing its usability and versatility. Users can retrieve real-time event data, manage events, and conveniently export event information in CSV format, all from a single platform.

(http://localhost:8080/AuleWeb/additional_pages/testcalendar.html?id=8&idAula=32)

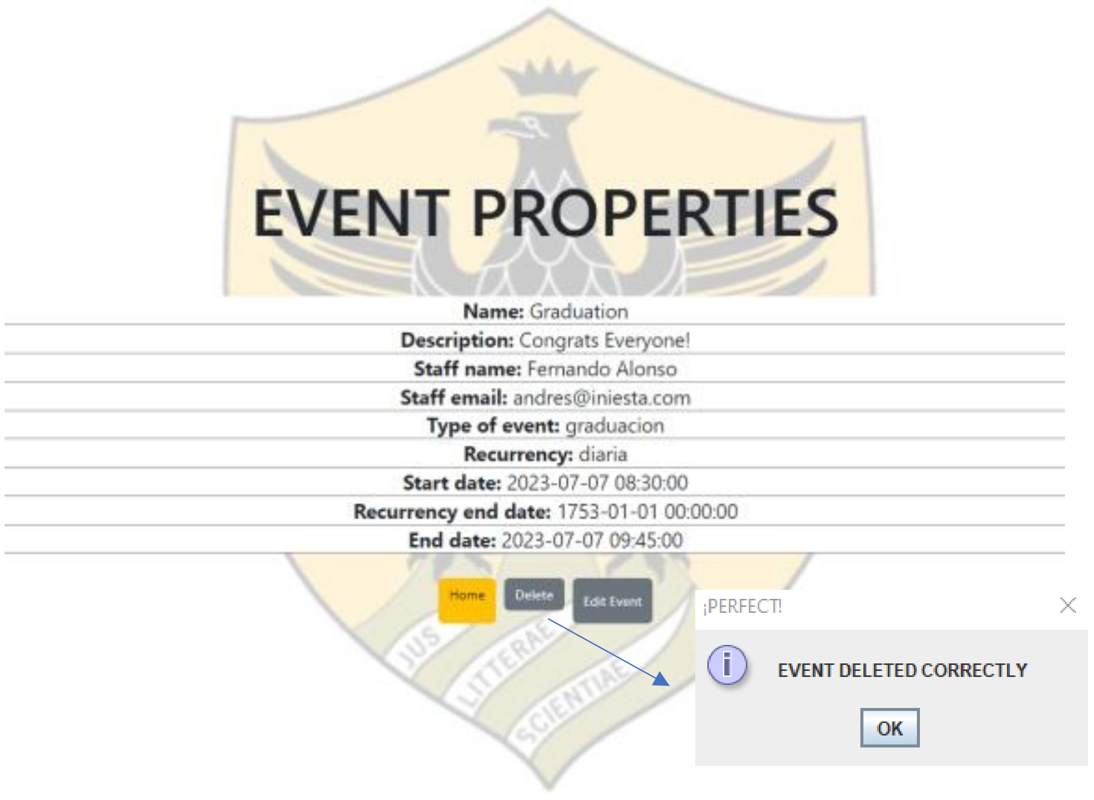


Room Name: VC 0.1
Room Description: d1
Room Capacity: 5
Number of Outlets: 50
Network Ports: 1
Has Projector: Yes
Has Motorized Screen: No
Has Manual Screen: Yes
Has Audio System: No
Has PC: Yes
Has Wireless Microphone: No
Has Wired Microphone: Yes
Has Overhead Projector: No
Has WIFI: No

When clicking on an event, the calendar page provides comprehensive information about that particular event. By selecting an event, users can access a detailed view that includes all relevant information associated with the event. This information may include the event's name, description, date, time, location, organizer, participants, and any other pertinent details.

By offering a detailed view of events, users can easily access and review specific event information without navigating to different pages or sources. This enhances the user experience by providing a centralized location for event management and access to event-related data.

(http://localhost:8080/AuleWeb/additional_pages/test.html?idDepartamento=8&idEvento=4&idEvento=4)



EVENT PROPERTIES

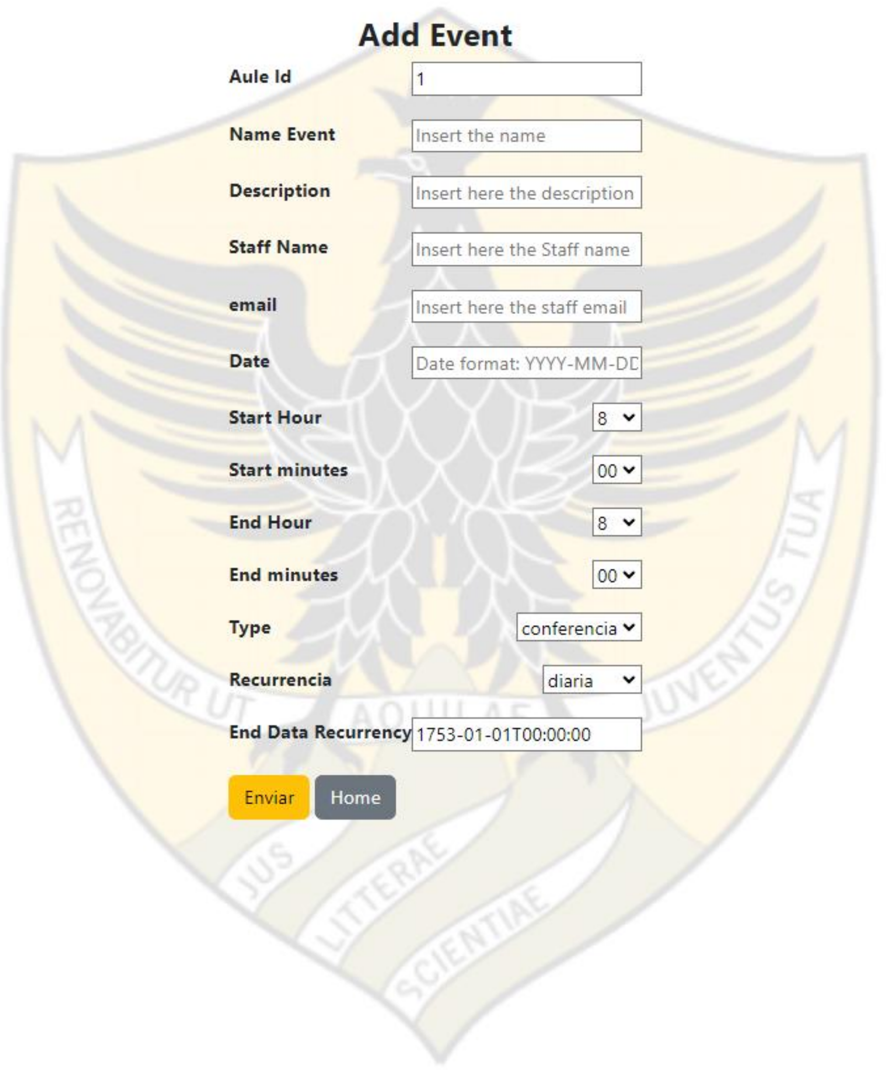
Name: Graduation
Description: Congrats Everyone!
Staff name: Fernando Alonso
Staff email: andres@iniesta.com
Type of event: graduacion
Recurrency: diaria
Start date: 2023-07-07 08:30:00
Recurrency end date: 1753-01-01 00:00:00
End date: 2023-07-07 09:45:00

¡PERFECT!

EVENT DELETED CORRECTLY

When selecting the Edit Event function, it redirects you to the page to add events, loading the information corresponding to the selected event.

(http://localhost:8080/AuleWeb/additional_pages/addevent.html?idDepartamento=8&idAula=28)



Add Event

Aule Id	1
Name Event	Insert the name
Description	Insert here the description
Staff Name	Insert here the Staff name
email	Insert here the staff email
Date	Date format: YYYY-MM-DD
Start Hour	8
Start minutes	00
End Hour	8
End minutes	00
Type	conferencia
Recurrencia	diaria
End Data Recurrency	1753-01-01T00:00:00

Implemented technologies

The technologies that we have used in the project are:

1. HTML5: HTML5 technology has been used for the frontend. Thanks to HTML, the client can see the data that is generated in the backend on his screen.
2. CSS: CSS technology has been used to format HTML. Thanks to CSS we can format the frontend and not just have plain text. Specifically, we have used bootstrap 5.3.0 to be able to generate the button, navbar, etc. styles automatically and thus give the web a unique and eye-catching design for the client.
3. JAVA: JAVA technology has been used for the creation of the entire backend. It is in charge of making the calls to the database, of obtaining all the data from the frontend through the doGET and doPOST requests offered by the Servlets. In addition, this technology has been used to perform validations.
4. JAVASCRIPT: JavaScript technology has been used to generate the skeleton of the frontend. Thanks to JS, we can get the data from the backend, loop through it, and display it on the frontend via HTML.
5. Version control: The GitHub application has been used to upload project updates. The project link is: https://github.com/lDrakohh/AuleWeb
6. Database: We have used HeidiSQL 12.5 (https://www.heidisql.com/download.php) to be able to visualize the data graphically. Also, in order to load the SQL server, we have installed XAMPP 8.2.4 (https://www.apachefriends.org/es/download.html).
7. IDE: Apache NetBeans IDE 15.

Mistakes descriptions

The list of errors (and that do not have to do with a bad syntax of a function) are:

1. Problems with jdbc dependency. When downloading the dependency and then making a call in the servlet, we were getting the error: "java.lang.ClassNotFoundException: com.mysql.cj.jdbc.Driver". This error was caused by file corruption. When we downloaded Tomcat Apache, a library was generated for us. After that, we install the XAMPP to be able to launch the MySql server. The problem comes when XAMPP automatically installs another Apache Tomcat again. This creates conflicts and causes the configuration and paths to get mixed up. When you read a dependency, sometimes it tries to detect it with the Tomcat Apache that was installed in the first place, and other times with XAMPP, causing the dependencies to not work.
2. Problems with ical4. Once the library has been downloaded and imported into the project. Apache netbeans didn't recognize the library at all. Several versions have been tried and it gave a reading error, so after several days trying to make it work, it was decided to use the "fullcalendar" library.
3. Problems with the gson dependency. We wanted to use this library to generate json format automatically, however we ran into a problem. When we loaded the library in a .java class, and ran it independently, the library worked. However, the problem arose when in the servlet we made a call to a function of the java class that used gson. The servlet gave the error: "java.lang.ClassNotFoundException: com.google.gson.Gson at org.apache.catalina", that is, from the Servlet it could not find the dependency as

installed. We tried looking for solutions on the internet but none worked for us, so we decided to manually parse the data as json.

4. Problems with the CSVWriter dependency. Basically, with this dependency, exactly the same thing happened to us as with the GSON library, so we had to generate the CSV manually.