**AuthController (Login & Sign-up Controller)**

**Class: AuthController**

This class is responsible for handling user authentication operations, including sign-up, login, and managing credentials and settings.

* **Route**: api/auth
* **Base Controller**: ControllerBase

**Constructor:**

csharp

Copy code

public AuthController(JTestCredentialsContext context)

* **Purpose**: Injects the JTestCredentialsContext for database operations.
* **Parameters**:
  + context: The database context used to interact with the application's database.

**Methods:**

1. **Signup**

csharp

Copy code

[HttpPost("signup")]

public async Task<IActionResult> Signup([FromBody] Credential credential)

* + **Purpose**: Handles the user sign-up by validating and saving new credentials.
  + **Parameters**:
    - credential: A Credential object containing the email and password for the new user.
  + **Logic**:
    - Validates the input (email and password).
    - Checks if the email is already taken.
    - Saves the new credential to the database if valid.
  + **Returns**:
    - BadRequest: If the credentials are invalid or the email is already taken.
    - Ok: If the user is successfully created.

1. **Login**

csharp

Copy code

[HttpPost("login")]

public async Task<IActionResult> Login([FromBody] Credential loginCredential)

* + **Purpose**: Handles user login by validating provided credentials against stored ones.
  + **Parameters**:
    - loginCredential: A Credential object containing the email and password.
  + **Logic**:
    - Validates the email and password.
    - Looks for a user with the provided email.
    - Compares the provided password with the stored password.
  + **Returns**:
    - Unauthorized: If the user is not found or password is invalid.
    - Ok: If the login is successful.

1. **Getdetails**

csharp

Copy code

[HttpGet("details")]

public async Task<IActionResult> Getdetails()

* + **Purpose**: Fetches all stored details from the database.
  + **Returns**:
    - Ok: Returns the list of stored details.

1. **GetCredentials**

csharp

Copy code

[HttpGet("credentials")]

public async Task<IActionResult> GetCredentials()

* + **Purpose**: Retrieves all stored credentials from the database.
  + **Returns**:
    - Ok: Returns the list of credentials.

1. **SaveCredentials**

csharp

Copy code

[HttpPost("save")]

public async Task<IActionResult> SaveCredentials([FromBody] Detail credential)

* + **Purpose**: Saves user credentials (such as domain, username, and API token) to the database.
  + **Parameters**:
    - credential: A Detail object containing credential data.
  + **Logic**:
    - Validates the credential fields.
    - Saves the credential to the database.
  + **Returns**:
    - BadRequest: If any field is missing.
    - Ok: If credentials are saved successfully.
    - StatusCode(500): If an error occurs during saving.

1. **DeleteDetail**

csharp

Copy code

[HttpDelete("detail/delete/{id}")]

public async Task<IActionResult> DeleteDetail(int id)

* + **Purpose**: Deletes a stored detail from the database by its ID.
  + **Parameters**:
    - id: The ID of the credential to be deleted.
  + **Logic**:
    - Looks for the credential by ID and deletes it.
  + **Returns**:
    - NotFound: If no credential is found with the specified ID.
    - Ok: If the credential is successfully deleted.

1. **DeleteCredential**

csharp

Copy code

[HttpDelete("credential/delete/{id}")]

public async Task<IActionResult> DeleteCredential(int id)

* + **Purpose**: Deletes a stored credential from the database by its ID.
  + **Parameters**:
    - id: The ID of the credential to be deleted.
  + **Logic**:
    - Looks for the credential by ID and deletes it.
  + **Returns**:
    - NotFound: If no credential is found with the specified ID.
    - Ok: If the credential is successfully deleted.

**UserStoryDescriptionController (User Story Controller)**

**Class: UserStoryDescriptionController**

This class provides endpoints for handling user stories, including generating test cases and exporting issues from Jira.

* **Route**: api/userstorydescription
* **Base Controller**: ControllerBase

**Constructor:**

csharp

Copy code

public UserStoryDescriptionController(ILogger<UserStoryDescriptionController> logger)

* **Purpose**: Injects the logger to log information, warnings, and errors.
* **Parameters**:
  + logger: Logger instance for logging messages.

**Methods:**

1. **GenerateTestCases**

csharp

Copy code

[HttpPost("GenerateTestCases")]

public async Task<IActionResult> GenerateTestCases([FromBody] UserStoryRequest request)

* + **Purpose**: Generates test cases based on the provided user story using the Gemini API.
  + **Parameters**:
    - request: A UserStoryRequest object containing the user story.
  + **Logic**:
    - Validates the user story format.
    - Calls the Gemini API to generate test cases for the user story.
  + **Returns**:
    - BadRequest: If the user story is invalid or empty.
    - Ok: If the test cases are successfully generated.

1. **IsValidUserStory**

csharp

Copy code

private bool IsValidUserStory(string userStory)

* + **Purpose**: Validates the user story format by checking if it contains common phrases.
  + **Parameters**:
    - userStory: The user story text.
  + **Returns**:
    - true: If the user story matches a valid structure.
    - false: Otherwise.

1. **GenerateTestCasesFromGemini**

csharp

Copy code

private async Task<string> GenerateTestCasesFromGemini(string userStory)

* + **Purpose**: Calls the Gemini API to generate test cases for the provided user story.
  + **Parameters**:
    - userStory: The user story text.
  + **Returns**:
    - The response content from Gemini API (test cases) if successful.
    - null: If the API request fails or an exception occurs.

1. **ExportAllUserStories**

csharp

Copy code

[HttpGet("ExportAllUserStories")]

public async Task<IActionResult> ExportAllUserStories([FromQuery] JiraRequestParams request)

* + **Purpose**: Retrieves user stories (Jira issues) based on the provided parameters from Jira.
  + **Parameters**:
    - request: A JiraRequestParams object containing the Jira domain, project, issue type, username, and API token.
  + **Logic**:
    - Validates the request parameters.
    - Makes a request to Jira to fetch issues based on the specified filters.
  + **Returns**:
    - BadRequest: If any required parameters are missing.
    - Ok: Returns the list of issues (user stories) from Jira.
    - StatusCode(500): If there’s an error fetching data from Jira.

**Conclusion:**

* Both controllers handle asynchronous operations using async and await for non-blocking calls to the database and external APIs.
* The controllers employ proper logging for tracking operations and errors, helping with troubleshooting and debugging.
* The AuthController deals with user credential management, while the UserStoryDescriptionController manages user stories and integrates with external APIs (Gemini and Jira).