# Backend Endpoints

### Projects

##### [GET] Projects/?pageSize={pageSize}&pageNumber={pageNumber}&{filter}={value}

projectService > *getProjectByFilter*(pageNumber, filter)

* **Purpose**: Gets projects by a given filter
* **Security**: Logged In
* **Url parameters**:
  + filter (String): the filters which you want the projects to be filtered by
    - Supports every projects property with equals, less (or equal) than, greater (or equal) than comparators (for example “Name == “SIT Project””)
    - Supports child properties (for example: “Lead.Id == “e980a9d8-53e5-4f6b-b8ae-1efec2e58938””)
    - Supports multiple criterias using “and” and “or” in between them (for example “Lead.Username == "admin@softuni.bg" or Description.Contains("test"))
  + pageSize (Int, Required): how many elements do you want the system to return
  + pageNumber (Int, Required): from which page to start (take the first pageSize \* pageNumber elements)
* **Returns**: The projects with their leaders

##### [GET] Projects/{id}

projectService > *getProjectById*(id)

* **Purpose**: Gets a project by id
* **Security**:Logged in

##### [POST] Projects/

* **Purpose**: Adds a new project with a default transition scheme which holds these statuses: Open, Closed, In Progress, Closed Progress
* **Security**: Admin
* **Body parameters**:
  + Name (Required)
  + Description (Required)
  + ProjectKey (Required): Should be generated by the first letters of the name. (Example: Project.Name = “Java Fundamentals” => Project.ProjectKey should be “JF”
  + List<Labels>: A list of labels which will be added to the project
  + List<Priorities>: A list of priorities which the project’s issues will be able to use
  + LeadId: The id of the person who will be leading the project
* **Returns**: The newly created project

##### [PUT] Projects/{id}

projectService > *editProject*(projectToEdit)

* **Purpose**: Edits a project by a given id
* **Security**:Admin, Lead of project
* **Body parameters**:
  + Same as **[POST] Projects/**, except for ProjectKey which cannot be edited
* **Returns**: The edited project

#### Issues

##### [GET] Projects/{id}/Issues

projectService > *getIssuesByProjectId*(id)

* **Purpose**: Gets the project’s issues by id
* **Security**:Logged in
* **Returns**: The project’s issues with their available statuses

##### [GET] Issues/?pageSize={pageSize}&pageNumber={pageNumber}&{filter}={value}

* **Purpose**: Gets issues by a given filter
* **Security**: Logged In
* **Url parameters**:
  + filter (String): the filters which you want the issues to be filtered by
    - Supports every issue’s property with equals, less (or equal) than, greater (or equal) than comparators (for example “ProjectId == 2”, “DueDate.Day >= 20”)
    - Supports child properties (as seen above: “DueDate.Day < 10”, “Project.Name” == “SIT”)
    - Supports multiple criterias using “and” and “or” in between them (for example “Priority.Name == "In Progress" or DueDate.Month == 3”)
  + pageSize (Int, Required): how many elements do you want the system to return
  + pageNumber (Int, Required): from which page to start (take the first pageSize \* pageNumber elements)
* **Returns**: The issues with their available statuses

##### [GET] Issues/me?pageSize={pageSize}&pageNumber={pageNumber}&orderBy={by}

* **Purpose**: Gets the user’s currently assigned issues ordered by a given criteria
* **Security**: Logged in
* **Url parameters**:
  + orderBy (String): the property of the issue which you want the issues to be sorted by
    - Supports all issue’s properties (for example Project, IssueKey, DueDate)
    - Supports child properties (for example Project.Name will sort the issues by the name of their project)
    - Supports descending sorting, just add “desc” after the property (for example “IssueKey desc”)
    - Supports multiple criteria using comma separated syntax (for example “Project.Name desc, IssueKey, Priority.Name desc”)
  + pageSize (Int, Required): how many elements do you want the system to return
  + pageNumber (Int, Required): from which page to start (take the first pageSize \* pageNumber elements)
* **Returns**: The user’s issues with their available statuses

##### [GET] Issues/{id}

issueService > getIssueById(id)

* **Purpose**: Gets an issue by id
* **Security**: Logged in
* **Returns**: The requested issue with its available statuses

##### [POST] Issues/

issueService > *createIssue*(issueToCreate)

* **Purpose**: Adds a new issue
* **Security**: Admin, Project lead
* **Body parameters**:
  + Title (String, Required)
  + Description (String, Required)
  + DueDate (DateTime, Required)
  + ProjectId (Int, Required): The issue’s project
  + AssigneeId (String, Required): The issue’s assignee
  + PriorityId (Int, Required): The priority’s id (should be one of the available priorities for the project)
  + List<Label> (Required): A list of labels for the issue
* **Returns**: The newly created issue

##### [PUT] Issues/{id}

issueService > *editIssueById*(id, issueToEdit)

* **Purpose**: Edits an issue by an id
* **Security**: Admin, Project lead
* **Body parameters**: Same as [POST] Issues/, except for ProjectId which cannot be edited
* **Returns**: The edited issue

##### [PUT] Issues/{id}/changestatus?statusid={statusId}

issueService > *changeIssueStatus*(issueId, statusId)

* **Purpose**: Edits an issue’s current status, only if it’s available in the status transition scheme (for example, you can’t change the status from ‘Open’ directly to ‘Stopped Progress’)
* **Security**: Admin, Issue assignee, Project lead
* **Url Parameters**:
  + statusid (Int, Required): the id of the new status
* **Returns**: The new available statuses

##### [GET] Issues/{id}/comments

issueService > *getAllComments*(id)

* **Purpose**: Gets all the issue’s comments by a specified id
* **Security**: Logged in

##### [PUT] Issues/{id}/comments

issueService > addComment(id, comment)

* **Purpose**: Adds a new comment to an issue specified by id
* **Security**: Logged in user who is either a project leader or has a assigned issue in this project
* **Body parameters**:
  + Text (String, Required): The comment’s text
* **Returns**: list of all the issue’s comments

#### Labels

##### [GET] Labels/?filter={filter}

* **Purpose**: Gets all of the existing labels filtered
* **Security**:Logged in
* **Returns**: The labels with their id and name
* **Url Parameters**:
  + filter (String, Required): The starting substring for the searched labels (For example: “?filter=sof” can return “Softuni, software” and every existing label that starts with “sof”)

#### Users

##### [POST] api/Account/Register

authenticationService > *registerUser*(user)

* **Purpose**: Registers a new, non-admin, user to the system
* **Security**: None
* **Body parameters**:
  + Email (String, Required): The email for the newly registered user
  + Password (String, Required): Password
  + ConfirmPassword: The same password for confirmation
* **Returns**: Status code 200 on success

##### [POST] api/Token

authenticationService > *loginUser*(loginData)

* **Purpose**: Gets an authentication token from the system to later authenticate the user (client) with the requests he makes
* **Security**: None
* **Body parameters**:
  + Username (String, Required): Use the user’s email as it serves as the username in the system
  + Password (String, Required): Password
  + grant\_type (String, Required): Should be always “password” in order to authenticate successfully
* **Returns**: Data with a field access\_token which is the necessary token needed for authentication
* **Usage**: After getting the token you can authenticate every request made by the client by putting an “Authorization” key in the request headers with value: “Bearer {access\_token}” where {access\_token} is the one returned from the system.

##### [GET] Users/me

identityService > loadCurrentUserData

* **Purpose**: Gets the currently
* **Security**:Logged in
* **Returns**: The user with his id, username and whether they’re admin

##### [POST] api/Account/ChangePassword

identityService >

* **Purpose**: Changes the current user’s password
* **Security**: Logged in
* **Body parameters**:
  + OldPassword (String, Required): The user’s current password
  + NewPassword (String, Required): New Password
  + ConfirmPassword (String, Required): Again the new password for confirmation
* **Returns**: Status code 200 on success

##### [PUT] Users/makeadmin

userService >

* **Purpose**: Grants an user admin privileges
* **Security**: Admin
* **Returns**: Status code **200** on success
* **Body parameters**:
  + UserId (String, Required): The id of the user to be made admin

##### [GET] Users/

userService > getAllUsers()

* **Purpose**: Gets all of the registered users
* **Security**:Logged in
* **Returns**: The users with their id, username and whether they’re admin

##### 