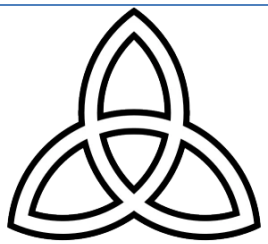
CPSC 3720 SPRING 2019

Web Service Documentation



Disir

Flores, Jonathan

December 01, 2019

Project Disir
Web Service Documentation

REST Endpoints

The server currently only supports two types of requests: POST Requests and GET Requests. All the create functions, update functions and delete functions use POST Request. Get list of objects functions and load/save functions use GET Request. We will discuss how to use each REST Endpoint in a moment. Also, it is important to mention that by the time this Web Service Documentation was written, the server is able to run just in localhost:1234, so all the REST Endpoint in this document will start with: 'http://localhost:1234/disirrest'

Create an Issue:

To create an issue, the web client needs to make a POST Request with the following URLData: createissue^TitleOfIssue^Description^Creator^Type^Priority^OS^Component

createissue is a static string. It always has to be at the beginning of the URLData because the server reads the operation at the beginning of the URLData, and then it does all the necessary work.

TitleOfIssue is a normal string with the title of the Issue. Example: "My computer does not start".

Description is normal string with the description of the Issue. In the server, Description is considered to be Comment 1, and it is added to the vector of comments of the Issue. The creator of the comment is set as the creator of the Issue .

Creator is a string with the format of an integer with the UserID of the creator of the Issue. If the UserID does not exist in the server, an error is returned as a JSON Object saying "The User ID for Creator does not exist".

Type is a string with the type of the Issue. There can be only 3 types: **Feature, Bug, and Task**. The web client must send the server one of those 3 types.

Priority: is a string with the priority of the Issue. There can be only 3 types of priority: **Low, Normal, and High**. The web client must send the server one of those 3.

OS is a string with the OS of the Issue. There can only be 3 types of OS: **Windows, Mac, and Linux**. The web client must send the server one of those 3.

Component is a string with the Component of the Issue. There can only be 2 types: **Hardware, and Software.** The web client must send the server one of those 2.

An example of the whole URL for Create an Issue using a POST Request:
http://localhost:1234/disirrest with URLData : **createissue^"My computer does not start"^"The Screen is black and it does not start "^"1"^"Bug"^"High"^"Windows"^"Hardware"**

If the information is valid, and the POST Request succeeded, a JSON Object is returned with the ID of the new Issue. Example of JSON Object returned = [{"issueID" : "1"}]

Project Disir
Web Service Documentation

Create a User:

To create a user, the web client needs to make a POST Request with the following URLData:
createuser^Username^UserType

createuser is a static string. It always has to be at the beginning of the URLData because the server reads the operation at the beginning of the URLData, and then it does all the necessary work.

Username is a string with the username of the new User. The username cannot contain spaces, and it has to be unique. If the username already exists in the Server, an error is going to be displayed.

UserType is a string with the type of User. There can only be 2 types of User: **Employee and Client**. The web client must send the server one of those 2.

An example of the whole URL for Create a User using a POST Request:
http://localhost:1234/disirrest with URLData : **createuser^"RickyB"^"Employee"**

If the information is valid, and the POST Request succeeded, a JSON Object is returned with the ID of the new User. Example of JSON Object returned = [{"userID" : "1"}]

Create a Comment:

To create a comment, the web client needs to make a POST Request with the following URLData:
createcomment^IssueID^CreatorID^CommentText

createcomment is a static string. It always has to be at the beginning of the URLData because the server reads the operation at the beginning of the URLData, and then it does all the necessary work.

IssueID is a string with the format of an integer with the ID of the Issue where the comment is going to be added. If the IssueID does not exist in the server, an error is returned as a JSON Object saying "The Issue does not exist".

CreatorID is a string with the format of an integer with the UserID of the creator of the Comment. If the UserID does not exist in the server, an error is returned as a JSON Object saying "The User ID for Creator does not exist".\

CommentText is a string containing the text for the comment. It does not have restrictions.

An example of the whole URL for Create a Comment using a POST Request:
http://localhost:1234/disirrest with URLData : **createcomment^"1"^"2"^"Did you check that the power cord is connected?"**

If the information is valid, and the POST Request succeeded, a JSON Object is returned with the ID of the Issued containing the Comment plus a string saying "Comment Added". Example of JSON Object returned = [{"commentID" : "issueID. Comment Added"}]

Project Disir
Web Service Documentation

Get List of Issues:

To get a list with all the issues in the Server, the web client needs to make a GET Request with the following URL: http://localhost:1234/disirrest?op=getissues

op is a parameter with the value of getissues. The Server reads the parameter from the GET Request and do all the work.

It returns a JSON Object with an array with all the issues in following format: [{ID: "issueID", title : "Title", description : "Description", statusIssue : "Status", assignedTo : "UserID", Creator : "UserID", typeIssue : "Type", priority : "Priority", os : "OS", component : "Component"}, {…}, ...]

Get an Issue:

To get a specific issue from the Server, the web client needs to make a GET Request with the following URL: http://localhost:1234/disirrest?op=getissues&id=IssueID

op is a parameter with the value of getissues. The Server reads the parameter from the GET Request and do all the work.

Id is a parameter with the value of the IssueID. If the IssueID does not exist, an error message is returned with the string "The Issue does not exist".

If the IssueID exists, a JSON Object is returned in the following format: [{ID: "issueID", title : "Title", description : "Description", statusIssue : "Status", assignedTo : "UserID", Creator : "UserID", typeIssue : "Type", priority : "Priority", os : "OS", component : "Component"}]

Get List of all Users:

To get a list with all the users in the Server, the web client needs to make a GET Request with the following URL: http://localhost:1234/disirrest?op=getusers

op is a parameter with the value of getusers. The Server reads the parameter from the GET Request and do all the work.

It returns a JSON Object with an array with all the users in following format: [{userID: "userID", userName : "Username", userType : "Type"}, {…}, ...]

Get a User:

To get a specific user from the Server, the web client needs to make a GET Request with the following URL: http://localhost:1234/disirrest?op=getusers&id=UserID

op is a parameter with the value of getusers. The Server reads the parameter from the GET Request and do all the work.

Id is a parameter with the value of the UserID. If the UserID does not exist, an error message is returned with the string “The User does not exist”.

It returns a JSON Object with the user in following format: [{userID: “userID”, userName : “Username”, userType : “Type”}]

Get a List of Comments from an Issue

To get a list with all the comments from an Issue in the Server, the web client needs to make a GET Request with the following URL: http://localhost:1234/disirrest?op=getcomments&issueID=IssueID

op is parameter with the value of getcomments. The Server reads the parameter from the GET Request and do all the work.

issueID is a parameter with the value of the IssueID. If the IssueID does not exist, an error message is returned with the string “The Issue does not exist”.

If the IssueID is valid, it returns a JSON Object with an array with all the comments from an Issue in following format: [{commentID: “commentID”, text : “Text of the Comment”, creator : “UserID”}, {…}, …]

Get a Comment from an Issue

To get a specific comment from an Issue in the Server, the web client needs to make a GET Request with the following URL:
http://localhost:1234/disirrest?op=getcomments&issueID=IssueID&commentID=CommentID

op is parameter with the value of getcomments. The Server reads the parameter from the GET Request and do all the work.

issueID is a parameter with the value of the IssueID. If the IssueID does not exist, an error message is returned with the string “The Issue does not exist”.

commentID is a parameter with the value of the CommentID. If the commentID does not exist, an error message is returned with the string “The Comment does not exist”.

If IssueID and CommentID are valid, it returns a JSON Object with the comment from an Issue in following format: [{commentID: “commentID”, text : “Text of the Comment”, creator : “UserID”}]

Update an Issue

To update an issue, the web client needs to make a POST Request with the following URLData:
updateissue^ID^TitleOfIssue^Description^Status^AssignedTo^Creator^Type^Priority^OS^Component

updateissue is a static string. It always has to be at the beginning of the URLData because the server reads the operation at the beginning of the URLData, and then it does all the necessary work.

Project Disir
Web Service Documentation

ID is the ID of the Issue. If the IssueID does not exists, an error messaged is returned as a JSON Object with the text “Issue does not exist”.
TitleOfIssue is a normal string with the title of the Issue. Example: “My computer does not start”.

Description is normal string with the description of the Issue. In the server, Description is considered to be Comment 1, and it is added to the vector of comments of the Issue. The creator of the comment is set as the creator of the Issue .

Status is a string with the status of the Issue. There can only be 4 status: **New, Assigned, Fixed, WontFix**. The web client must send the server one of those 4 status. By default, A New status is assigned to all new Issues.

AssignedTo is a string with the format of an integer with the UserID of the employee assigned to the Issue. If the UserID does not exist in the server, an error is returned as a JSON Object saying “The User ID for AssignedTo does not exist”.

Creator is a string with the format of an integer with the UserID of the creator of the Issue. If the UserID does not exist in the server, an error is returned as a JSON Object saying “The User ID for Creator does not exist”.

Type is a string with the type of the Issue. There can be only 3 types: **Feature, Bug, and Task**. The web client must send the server one of those 3 types.

Priority: is a string with the priority of the Issue. There can be only 3 types of priority: **Low, Normal, and High**. The web client must send the server one of those 3.

OS is a string with the OS of the Issue. There can only be 3 types of OS: **Windows, Mac, and Linux**. The web client must send the server one of those 3.

Component is a string with the Component of the Issue. There can only be 2 types: **Hardware, and Software.** The web client must send the server one of those 2.

An example of the whole URL for Update an Issue using a POST Request:
http://localhost:1234/disirrest with URLData : **updateissue^”1”^”My computer does not start”^”The Screen is black and it does not start ”^”Assigned”^”3”^”1”^”Bug”^”High”^”Windows”^”Hardware”**

If the information is valid, and the POST Request succeeded, a JSON Object is returned with the string “Issue Updated”. Example of JSON Object returned = [{“Status” : “Issue Updated”}]

Update a User:

To update a user, the web client needs to make a POST Request with the following URLData:
updateuser^Id^Username^UserType

updateuser is a static string. It always has to be at the beginning of the URLData because the server reads the operation at the beginning of the URLData, and then it does all the necessary work.
Id is the UserID. If UserID does not exist, an error message is returned as a JSON Object with the string “User does not exist”.

Username is a string with the username of the new User. The username cannot contain spaces, and it has to be unique. If the username already exists in the Server, an error is going to be displayed.

UserType is a string with the type of User. There can only be 2 types of User: **Employee and Client**. The web client must send the server one of those 2.

An example of the whole URL for Update a User using a POST Request:
http://localhost:1234/disirrest with URLData : **updateuser^1^”RickyB”^”Client”**

If the information is valid, and the POST Request succeeded, a JSON Object is returned with the string “User Updated”. Example of JSON Object returned = [{Status : “User Updated”}]

Update a Comment:

To update a comment, the web client needs to make a POST Request with the following URLData:
updatecomment^IssueID^CommentID^CreatorID^CommentText

updatecomment is a static string. It always has to be at the beginning of the URLData because the server reads the operation at the beginning of the URLData, and then it does all the necessary work.

IssueID is a string with the format of an integer with the ID of the Issue where the comment has been added. If the IssueID does not exist, an error is returned as a JSON Object saying “The Issue does not exist”.

CommentID is a string with the format of an integer with the ID of the Comment. If the CommentID does not exist, an error is returned as a JSON Object saying “The Comment does not exist”.

CreatorID is a string with the format of an integer with the UserID of the creator of the Comment. If the UserID does not exist i, an error is returned as a JSON Object saying “The User ID for Creator does not exist”.

CommentText is a string containing the text for the comment. It does not have restrictions.

An example of the whole URL for Update a Comment using a POST Request:
http://localhost:1234/disirrest with URLData : **updatecomment^”1”^”1”^”2”^”Did you check that the power cord is connected? Is the monitor on?”**

If the information is valid, and the POST Request succeeded, a JSON Object is returned with the string “Comment Updated”r. Example of JSON Object returned = [{Status : “Comment Updated”}]

Project Disir
Web Service Documentation

Delete an Issue

To delete an issue, the web client needs to make a POST Request with the following URLData:
delete^deleteissue^ID

delete is a static string. It always has to be at the beginning of the URLData because the server reads the operation at the beginning of the URLData, and then it does all the necessary work.

deleteissue is a static string. The server needs this part to know which delete operation has to be done.

ID is a string with the format of an integer with the IssueID. The Server needs the IssueID to find the Issue and remove it internally. If the IssueID does not exist, an error message is returned as a JSON Object with the string “Issue does not exist”.

If IssueID is valid, the Server deletes the Issue and returns a JSON Object with the string “Issue Deleted”. Example of JSON Object returned: [{Status : “Issue Deleted”]

Delete a User

To delete a User, the web client needs to make a POST Request with the following URLData:
delete^deleteuser^ID

delete is a static string. It always has to be at the beginning of the URLData because the server reads the operation at the beginning of the URLData, and then it does all the necessary work.

deleteuser is a static string. The server needs this part to know which delete operation has to be done.

ID is a string with the format of an integer with the UserID. The Server needs the UserID to find the User and remove it internally. If the UserID does not exist, an error message is returned as a JSON Object with the string “User does not exist”.

If UserID is valid, the Server deletes the User and returns a JSON Object with the string “User Deleted”. Example of JSON Object returned: [{Status : “User Deleted”]

Delete a Comment

To delete a comment from an issue, the web client needs to make a POST Request with the following URLData:
delete^deletecomment^IssueID^CommentID

delete is a static string. It always has to be at the beginning of the URLData because the server reads the operation at the beginning of the URLData, and then it does all the necessary work.

deletecomment is a static string. The server needs this part to know which delete operation has to be done.

Project Disir
Web Service Documentation

IssueID is a string with the format of an integer with the ID of the Issue. The Server needs the IssueID to find the Issue that contains the Comment that wants to be deleted. If the IssueID does not exist, an error message is returned as a JSON Object with the string “Issue does not exist”.

CommentID is the ID of the Comment. The Server needs the CommentID to find the Comment among the Comments from the Issue. If the CommentID does not exist, an error message is returned as a JSON Object with the string “Comment does not exist”.

If IssueID and CommentID are valid, the Server deletes the comment and returns a JSON Object with the string “Comment Deleted”. Example of JSON Object returned: [{Status : “Comment Deleted”]

Save State of Server

To save the state of the Server, the web client needs to make a GET Request with the following URL: http://localhost:1234/disirrest?op=saveserver&filename=NameOfTheFile

op is a parameter with the value of saveserver. The Server reads the parameter from the GET Request and do all the work.

Filename is a parameter with the value of the name of the file where the user wants to save the state of the file. The extension of the file has to be ‘.dat’.

If there are not errors while saving the state of the Server, a JSON Object is returned with the string “Server Saved”. Example of JSON Object returned: {Status: “Server Saved”}

Load a Server

To load a Server, the web client needs to make a GET Request with the following URL: http://localhost:1234/disirrest?op=loadserver&filename=NameOfTheFile

op is a parameter with the value of loadserver. The Server reads the parameter from the GET Request and do all the work.

Filename is a parameter with the value of the name of the file where a Server has been saved. The extension of the file has to be ‘.dat’. If the file does not exist, a JSON object is returned with the string “File not Found”.

If there are not errors while loading the state of the Server, a JSON Object is returned with the string “Server Loaded”. Example of JSON Object returned: {Status: “Server Loaded”}

Search By Priority

To search by priority, the web client needs to make a GET Request with the following URL: http://localhost:1234/disirrest?op=searchpriority&priority=PriorityOfIssue

op is a parameter with the value of searchpriority. The Server reads the parameter from the GET Request and do all the work.

priority is a parameter with a string with the priority of the Issue. There can be only 3 types of priority: **Low, Normal, and High**. The web client must send the server one of those 3.

If there are Issues with the same priority as the priority parameter, the Server returns a JSON Object with an array with all the issues with the specified priority in following format: [{ID: "issueID", title : "Title", description : "Description", statusIssue : "Status", assignedTo : "UserID", Creator : "UserID", typeIssue : "Type", priority : "Priority", os : "OS", component : "Component"}, {…}, …]

Search By Status

To search by status, the web client needs to make a GET Request with the following URL: http://localhost:1234/disirrest?op=searchstatus&status=StatusOfIssue

op is a parameter with the value of searchstatus. The Server reads the parameter from the GET Request and do all the work.

status is a parameter with a string with the status of the Issue. There can be only 4 types of status: **New, Assigned, Fixed, and WontFix**. The web client must send the server one of those 4.

If there are Issues with the same status as the status parameter, the Server returns a JSON Object with an array with all the issues with the specified status in following format: [{ID: "issueID", title : "Title", description : "Description", statusIssue : "Status", assignedTo : "UserID", Creator : "UserID", typeIssue : "Type", priority : "Priority", os : "OS", component : "Component"}, {…}, …]

Search By Type

To search by type, the web client needs to make a GET Request with the following URL: http://localhost:1234/disirrest?op=searchtype&type=TypeOfIssue

op is a parameter with the value of searchtype. The Server reads the parameter from the GET Request and do all the work.

type is a parameter with a string with the type of the Issue. There can be only 4 types of status: **New, Assigned, Fixed, and WontFix**. The web client must send the server one of those 4.

If there are Issues with the same type as the type parameter, the Server returns a JSON Object with an array with all the issues with the specified type in following format: [{ID: "issueID", title : "Title", description : "Description", statusIssue : "Status", assignedTo : "UserID", Creator : "UserID", typeIssue : "Type", priority : "Priority", os : "OS", component : "Component"}, {…}, ...]