Server REST API Pre and Post Conditions

This document defines the pre and post conditions for the server REST API calls.

Non-Move APIs

/user/login

Description:
Logs an existing user into the game server

Pre-conditions: None

Post-conditions:
If the passed-in (username, password) pair is valid,

1. The server returns an HTTP 200 success response with an empty body.
2. The HTTP response headers set the **catan.user** cookie to contain the identity of the logged-in player. The cookie uses "Path=/", and its value contains a url-encoded JSON object of the following form: { "name": STRING, "playerID": INTEGER }. For example, { "name": "Rick", "playerID": 14 }.

If the passed-in (username, password) pair is not valid,

1. The server returns an HTTP 400 error response. The response headers include "Content-Type: text/plain", and the body contains an error message that will be displayed to the user.

Notes:
The passed-in username and password may correspond to the credentials of any registered user. We start the server with four users: Sam, Brooke, Pete and Mark. Their passwords are sam, brooke, pete and mark respectively. Any additional registered users should also work with this call.

/user/register

Description:
This call does two things:
1) Creates a new user account
2) Logs the new user into the game server

Pre-conditions: None

Post-conditions:

If there is no existing user with the specified name,

1. The server creates a new user account with the specified username and password.
2. The server returns an HTTP 200 success response with an empty body.
3. The HTTP response headers set the **catan.user** cookie to contain the identity of the logged-in player. The cookie uses ”Path=/”, and its value contains a url-encoded JSON object of the following form: { “name”: STRING, “playerID”: INTEGER }. For example, { “name”: “Rick”, “playerID”: 14 }.

If there is already an existing user with the specified name,

1. The server returns an HTTP 400 error response. The response headers include “Content-Type: text/plain”, and the body contains an error message that will be displayed to the user.

Notes:

You should be able to register any username via this call, unless that username is already registered with another user. There is no changing passwords.

/games/list

Description:

This call returns information about all of the current games.

Pre-conditions: None

Post-conditions:

1. The server returns an HTTP 200 success response.
2. The response headers include “Content-Type: application/json”
3. The return format follows the specified JSON format
4. The id’s are integers, and the color is one of the 9 following values:
   red, green, blue, yellow, puce, brown, white, purple, orange

Output JSON format:

```
[
  {
    "title": "Game name",
    "id": 0,
    "players": [
      {
        "color": "orange",
        "name": "playerName",
        "id": 0
      },...
    ]
  },...
]
```

/games/create

Description:

Creates an empty game on the server.

Pre-conditions: **None**

Post-conditions:

1. The server returns an HTTP 200 success response.
2. The response headers include “Content-Type: application/json”
3. The return format follows the specified JSON format
4. The returned game has no players in the array, or only empty player objects

Output JSON format:

```
{
  "title": "Game Name",
  "id": 3,
  "players": [
    {},
    {},
    {},
    {}
  ]
}
```

/games/join

Description:

Adds a player to the game and sets their game cookie

Pre-conditions:

1. The player has a valid catan.user cookie set.
2. The player may join the game because
   2.a They are already in the game
   2.b There is space to add them as a new player
3. The color submitted is a valid one
   (red, green, blue, yellow, puce, brown, white, purple, orange)

Post-conditions:

1. The server returns an HTTP 200 success response.
2. The player is in the game, with the color specified.
   (ie. calls to games/list should show that player in the game, with the updated color)

3. The server response includes the “Set-cookie” reponse header, setting catan.game to be the game id, with path=”/”;

/game/model?version=x

Description:
Fetches the JSON for the model of the current game.

Pre-conditions:
1. The player has a valid catan.user and catan.game id

Post-conditions:
1. The server returns an HTTP 200 success response.
2. The response headers include “Content-Type: application/json”
3. The JSON is specified according to x:
    - 3.a If ?version=x is omitted, return the model JSON
    - 3.b If x is different from the current model.revision, return the model JSON
    - 3.c If x is the same as the model.revision, return “true” (true in quote marks)

Notes:
The value ‘x’ should be grabbed from the last model you got (i.e., model.version) - it will be an integer. If you omit the version number, you’ll still get the model. However, if you use it and the server detects that it’s from the latest model it will return “true” instead of the model.

/game/reset

Description:
Resets the game to how it was after all the players joined

Pre-conditions:
1. The player has a valid catan.user and catan.game id

Post-conditions:
1. The server returns an HTTP 200 success response.
2. The response headers include “Content-Type: application/json”
3. Returns the JSON for the game model

Notes:
The server should save a game after all the players have been added so that it can revert to that state.

/game/commands [GET]

Description:
Gets a list of all the commands played on a game.

Pre-conditions:

1. The player has a valid catan.user and catan.game id

Post-conditions:

1. The server returns an HTTP 200 success response.
2. The response headers include “Content-Type: application/json”
3. Returns the commands that have been executed in the game in a JSON list - they should be serialized such that they could be sent to the server again either in /game/commands or individually on their corresponding API endpoints.

/game/commands [POST]

Description:
Applies a list of commands to the current game.

Pre-conditions:

1. The player has a valid catan.user and catan.game id

Post-conditions:
If the content could not be deserialized, or an error in applying the commands:

1. The server returns an HTTP 400 success response.
2. The response is an error message detailing what failed.

If the commands were successfully deserialized:

1. The server returns an HTTP 200 success response.
2. The response headers include “Content-Type: application/json”
3. Returns the JSON for the game model.

/game/listAI

Description:
Lists the available AI types that may be added to a game

Pre-conditions:None

Post-conditions:

1. The server returns an HTTP 200 success response.
2. The response headers include “Content-Type: application/json”
3. Returns a JSON list of strings - each string is an AI name that can be used for /game/addAI

/game/addAI

Description:

Adds an AI to the game

Pre-conditions:

1. The player has a valid catan.user and catan.game id
2. There is space in the game for an AI player

Post-conditions:

1. The server returns an HTTP 200 success response.
2. The AI player is added to the next open spot in the game in the poster's catan.game cookie
3. The AI player uses a color not taken by any other player

/util/changeLogLevel

Description:

Sets the server's logging level

Pre-conditions:

1.The poster specifies a valid logging level. Valid values include: SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST

Post-conditions:

1. The server returns an HTTP 200 success response.
2. The Server uses that logging level

Move APIs

This section lists all the moves that one can make in a game. They are listed by the type string in the command (Commands are sent to the path '/moves/x' where 'x' is the type of the command).

We have listed the pre- and post-conditions for all of the moves. They are grouped in sections that contain common preconditions (like it being your turn.) If preconditions on a specific move are omitted, the only preconditions it has are are the ones from its section.

Additionally, possible values for arguments are listed here. We did not specify values if they are obvious from the property's name (i.e. *playerIndex* must be a **playerIndex**, and *resource1* must be a **Resource**, etc.)

Data Types:
playerIndex - An integer representing a player's position in the game's turn order. It has one of the following values: 0,1, 2, or 3.
EdgeLocation - Represents the location of an edge on the game map. The x and y properties are integers in the range [-3, 3] that represent a hex location. The direction property is one of [NW, N, NE, SE, S, SW].
VertexLocation - Represents the location of a vertext on the game map. The x and y properties are integers in the range [-3, 3] that represent a hex location. The direction property is one of [NW, NE, E, SE, SW, W]
Resource - A resource type which is one of [Brick, Wood, Sheep, Wheat, Ore]
ResourceHand - Represents a collection of resources cards. It has five properties: [Brick, Wood, Sheep, Wheat, Ore]. Each has an integer value. 0 means the card amount doesn't change. A negative number means you're gaining a card. Positive means you're giving a way a card.

<u>Any-time commands:</u>

1. sendChat
    a. Values:
        - *content*: string [the message you want to send]
    b. Precondition: none
    c. Postcondition
        - The chat contains your message at the end

<u>Misc Commands:</u>

2. acceptTrade
    a. Values
        - *willAccept*: boolean
    b. Preconditions
        - You have been offered a domestic trade
        - For accept: you have the resources
    c. Postconditions
        - If you accepted, you and the player who offered swap the specified resouces
        - If you declined no resources are changed
        - The trade offer is removed

3. discardCards
    a. Values
        - *discardedCards*: **ResourceHand** [the cards you're discarding]
    b. Preconditions
        - The state of the client model is 'Discarding'
        - You have over 7 cards
        - You have the cards you're choosing to discard.
    c. Postconditions
        - If you're the last one to discard, the client model status changes to 'Robbing'
        - You give up the specified resources

<u>Moves for your turn:</u>

4. rollNumber
    a. Values:
        - *number*: integer, 2-12 [the number you're rolling]
    b. Precondition
        - The client model's status is 'rolling'
        - It's your turn
    c. Postconditions
        - The client model's status is now in 'discarding' or 'robbing' or 'playing'

<u>'Playing' only commands:</u>

General preconditions:

- It's your turn
- The client model status is 'Playing'

5. buildRoad
    a. Values:
        - *free*: boolean [whether or not you get this piece for free (i.e., in setup)]
        - *roadLocation*: **EdgeLocation**
    b. Preconditions:
        - The road location is open
        - The road location is connected to another road
        - The road location is not on water
        - You have the resources (1 wood, 1 brick; 1 road)
    c. Postconditions:
        - You expend the resources to play the road (1 wood, 1 brick; 1 road)
        - The map lists the road correctly

6. buildSettlement
    a. Values:
        - *free*: boolean [whether or not you get this piece for free (i.e. in setup)]
    b. Preconditions:
        - The settlement location is open
        - The settlement location is not on water
        - The settlement location is connected to one of your roads
        - You have the resources (1 wood, 1 brick, 1 wheat, 1 sheep; 1 settlement)
    c. Postconditions:
        - You expend the resources to play the settlement (1 wood, 1 brick, 1 wheat, 1 sheep; 1 settlement)
        - The map lists the settlement correctly

7. buildCity

    a. Preconditions:
        - The city location is where you currently have a settlement
        - You have the resources (2 wheat, 3 ore; 1 city)
    b. Postconditions:
        - You expend the resources to play the settlement (2 wheat, 3 ore; 1 city)
        - You get a settlement back
        - The map lists the city correctly

8. offerTrade
    a. Values:
        - *offer*: **ResourceHand** [negative numbers mean you get those cards]
        - *receiver*: **playerIndex** [the recipient]
    b. Preconditions
        - You have the resources
    c. Postconditions
        - The trade is offered to the other player (stored in the model)

9. maritimeTrade
    a. Values:
        - *ratio*: **integer** [2,3 or 4]
        - *inputResource*: **Resource** What you are giving
        - *outputResource*: **Resource** What you are getting
    b. Preconditions
        - You have the resources
    c. Postconditions
        - The trade is offered to the other player (stored in the model)

10. finishTurn
    a. Precondition
        - The client model status is 'Playing'
    b. Postcondition
        - It’s the next players turn

11. buyDevCard
    a. Preconditions
        - You have the resources (1 ore, 1 wheat, 1 sheep)
        - There are dev cards left in the deck
    b. Postconditions
        - You have the new card
            - If it is a monument card, it goes into the old devcard hand
            - If it’s any other card, it goes into the new devcard hand (unplayable this turn)

Dev Cards

General Preconditions:

- You have the specific card you want to play in your 'old dev card hand'
- You haven't played a dev card this turn yet.
- It's your turn
- The client model status is 'Playing'

12. Year_of_Plenty
    a. Preconditions:
        - The two resources you specify are in the bank
    b. Postconditions:
        - You gain the two resources specified

13. Road_Building
    a. Values:
        - *spot1*, *spot2*: **EdgeLocation**
    b. Preconditions
        - The first road location is connected to one of your roads.
        - The second road location is connected to one of your roads or the Previous location
        - Neither location is on water
        - You have two roads
    c. Postconditions
        - You use two roads
        - The map lists the roads correctly

14. Soldier
    a. Values
        - *location*: **HexLocation**
        - *victimIndex*: **playerIndex** [who you're robbing]
    b. Preconditions
        - The robber isn't being kept in the same place
        - The player to rob has cards (-1 if you can't rob anyone)
    c. Postconditions
        - The robber is in the new location
        - The player to rob gives one random resource card to the player playing the soldier

15. Monopoly
    a. Preconditions : **None** (besides the general devcard ones)
    b. Postconditions
        - All other players lose the resource card type chosen
        - The player of the card gets an equal number

16. Monument
    a. Preconditions : **None** (besides the general devcard ones)

b. Postconditions
    - You gain a victory point