Part1

1A User cases：

Case 1:

Title: Start a tic-tac-toe game
Actor:: User(Player), Game Server(process the user's request and update the game)
Description: the starting part of tic-tac-toe game. Any player can start the game and pick his/her symbol in this game. The game will save the first player's information and wait for the second player. Updated views will be available to current players afterwards.
Preconditions: the game is available(the server is running)
Postconditions:the game has started with updated view and a link is provided for second player to join this game
Triggers: some player invokes play-game request
Basic Flow:

1. The player connects to the server and sees the gameboard page on the webpage.
2. The player choose either 'X' or 'O' character in this game
3. The player press button "start Game"
4. The server receives the request from the player and saves his/her information as first player.It create a new game and update the view for this player and provide link for second player to join in

Case 2:
Title: Join a tic-tac-toe game
Actors: first player, second player, game server(process the user's request and update the game)
Description: If a game has initialized with one player, any other player can use a join-game link to join this game and start playing tic-tac-toe with another player.
Preconditions: Some game has started and the join-game link has been provided to this second player
Postconditions: The game is updated with two players and player 1 is allowed to play first move
Triggers: the second player invokes join-game request using the join-game link
Basic Flow:

1. The second uses the join-game link provided by first player to send join-game request
2. The server receives the request and updates second player's information of this game and update the views for both players
3. Both players can see the updated game board and the first player is ready to make first move

Case 3:
Title: Player makes a move in tic-tac-toe game
Actor: some player in a tic-tac-toe game, the server which is running the game
Description:in a tic-tac-toe game, if it is the turn for some player, he/she can make a move in the game if such move is valid
Preconditions:the tic-tac-toe game has been initialized with 2 players in this game
Postconditions: The player successfully makes a move in the game and the gameboard has been updated for both players
Basic Flow:

1. The player pick a spot on the gameboard and click this spot to send request of making move on this spot(on UI)
2. The server receives the move requisition of this player and check this move's validity (no winner, no draw, the spot has not been taken)
3. The move has been checked and it is valid. The server updates its internal gameboard information and send successful messages back to the player who makes this move
4. The game board is updated by the server and the new view is updated for both players

Alternative flow:

3A1:

1. The move has been checked and it is invalid(either draw, win or spot taken). The server send back 'move invalid' along with error code to the player
2. This player's gameboard page has been updated with 'move invalid' notification and his move is not made
3. This use case goes back to 1st step of basic flow

Exception Flow:

1. The game is in draw or some player has won the game or spot taken. The players cannot make any move without restarting the game

Case 4:
Title: one player wins tic-tac-toe game
Actors: two players in tic-tac-toe game and the server which is running this game
Description: In a tic-tac-toe game, one player can win this game
Preconditions:the tic-tac-toe game has been initialized with 2 players in this game
Postconditions: One player successfully wins the game by making his last move. The view is updated for both players showing that this player has won the game. The game stops and no one can make any more moves.
Triggers: some player makes a move that can make him/her win
Basic Flow:

1. Some player in a game makes a move on gameboard(on UI)
2. The server receives the move-request
3. The server checks this move's validity and it is valid, updating the game information.
4. The server checks if this game has a winner. It finds out the player who just made the move is the winner.

5. The server updates the game board and sends back a ‘move successful’ message to the player. The new view is updated for both players with “player X wins the game” as the message. No one can make any more moves

Case 5:
Title: The game is a draw
Actors: two players in a tic-tac-toe game and the server which is running the game
Description: in a tic-tac-toe game, if all moves are exhausted and no one has won the game, then it is a draw
Preconditions:the tic-tac-toe game has been initialized with 2 players in this game
Postconditions: the tic-tac-toe game ends with draw messages sent to both players
Triggers: all moves are exhausted and no one wins
Basic Flow:

1. Some player in a game makes last move on game board(on UI)
2. The server receives the move-request and check the validity of the move
3. The move is valid and the gameboard is updated by the server
4. The server checks the status of the server and finds out it is a draw situation. The server sends a “draw” message back to the players and the view is updated for both players. No more move can be made

Case 6:
Title: the game crashed and recovered
Actors: server. 0 players or 1 or 2 players
Description: tic-tac-toe game crashes and it should be able to restart with the last saved state of game
Postconditions: after the game restart, all states should be recovered and players should be able to resume the game
Triggers: application crashes
Basic Flow:

1. The application crashes
2. The application recovers with last saved state
3. The application resumes with last saved state:
    a. If no game started, the application restart with no game
    b. The game started. The application should restart with a game shown in view for players
    c. The game had 2 players then the restarted state should include 2 players’ information
    d. The game board includes all moves of all players after the application recovers
    e. If the game was a draw or win situation, after the game recovers, the game should be also in draw/win situation

1B:

Major methods:

PlayGame Class:

- Main: this method is used for updating gameboard and sending messages to players
  - Use Case 1:
    - Basic flow steps: 1, 2, 3, 4
    - Postconditions and preconditions
  - Use Case 2:
    - Preconditions and postconditions
    - Basic Flow steps: 1, 2, 3
  - Use Case 3:
    - Preconditions and Postconditions
    - Basic Flow Steps: 1, 2, 4
    - Alternative flow: 2, 3
  - Use Case 4:
    - Preconditions and Postcondition
    - Basic Flow Steps: 1, 2,5
  - Use Case 5
    - Preconditions and postconditions
    - Basic Flow steps: 1, 2
  - Use Case 6
    - Preconditions and postconditions
    - Basic Flow Steps: 1, 2, 3,

Gameboard Class:

- addMove(Move move):
  - Use Case 3:
    - Basic Flow: 2, 3
    - Alternative flow: 1
    - Exception flow: 1
  - Use Case 4:
    - Basic flow: 3
  - Use case 5:
    - Basic flow: 3,

- checkWinner():
  - Use Case 3:
    - Basic Flow: 2
    - Alternative flow: 1
    - Exception flow: 1

Use Case 4:
- Basic flow: 3, 4

Use Case 5:
- Basic Flow: 2

checkChar(char c)(helper method for checkWinner):

Use Case 3:
- Basic Flow: 2
- Alternative flow: 1
- Exception flow: 1

Use Case 4:
- Basic flow: 3, 4

Use Case 5:
- Basic Flow: 2

checkMove(Move move): this method checks for invalid move, isdraw or winner situation

Use Case 3:
- Basic Flow: 2
- Alternative Flow: 1
- Exception Flow : 1

Use Case 4:
- Basic Flow: 3, 4

Use Case 5:
- Basic Flow: 2, 3, 4

restoreState(connection conn):

Use Case 6:
- Basic flow: 2, 3

1C:

I think after 1A , 1B I have learnt that my codes did not divide functionality well. There are chaining issues within my code. If I have the chance to redesign my code, I think I am creating more major methods to separate functionalities better so that each functionality is satisfied by a single major method. Also, I would have reduced the chaining in my code as well.

Part2:

2A: APIs:

Method 1:
REST-Method: GET
Endpoint : /meeting
e.g https://api.zoom.us/v2/meeting
Description: get the ids of all soli chips that are working in a meeting
Request Parameters:
Id: the id of the meeting
Responses: the list of IDs of all soli chips that are working in this meeting specified by the given id of the meeting

Method 2:
REST-Method: GET
Endpoint: /soli/
Descriptions: return the state of the user recorded in the soli chip
Request Parameters:
Id: the id of the soli chip
Responses: the state of the user recorded in the soli chip which includes the chip's running time, the user's behaviors or activities recorded

Method 3:
Rest-Method: GET
Endpoint: /soli/{id}/user
Path Parameters:
Id: the id of the soli chip
Responses: the chip's user's information which includes names, columbia uni.

2B:

Story 1: As an instructor, I want to spot students who are not paying attention during the class so that I can notify him/her to be more active in class

Conditions of satisfaction:
1. User must be authorized with valid token as an Columbia Instructor
2. User's access level is classified as instructor(can access recorded data)
3. The meeting is classified as Columbia class
4. User is notified in real-time if some students are not paying attention

5. The User will receive full information includes name/uni of the students not paying attention
6. The user is provided with functionality to notify those students who do not pay attention

Story 2: As an student, I want to be notified if I were not paying attention so that I can learn more in class

Conditions of satisfaction:

1. User must be a participant in Columbia zoom Class(authorized)
2. User had agreed to terms and conditions of privacy that allows soli chip to record
3. User's info must be authenticated as a student of columbia university
4. User's access level is classified as student(cannot access recorded data, cannot notify others)

Story 3: As an observer, I want to monitor how well students are during class so that I can report the performance of the instructor

Conditions of satisfaction:

1. User must be a participant in Columbia Zoom Class
2. User must be authenticated as an observer assigned by Columbia University
3. User's level is classified as observer(can access recorded data, cannot notify others)
4. User's information which includes uni/names is kept hidden from any other member of the meeting

2C: I would simulate a meeting along with some users. The acceptance test will test if user with instructor access can access the data of students and verify the validity of those data. And check if the data of soli chip can be retrieved from the chip through REST API and the data is not corrupted

2D: I would simulate a meeting with users of one of the 3 roles: instructors, observer, and students. Some students would pretend not to pay attention but while doing behaviors to confuse the soli chip and some students would pretend to pay attention. Based on the result of how many students are misclassified, we conclude the acceptance test