Design of Covey.Town project

Backend

We modified the backend of Covey.Town so that it supports users and invitations.

Introduction

We added services related to user, invitation and tried to keep all the original designs. Now the join town pipeline becomes:

1. create username => select a town to join / create a town => entering town
2. use an invitation link => create username => entering town

Therefore, we added classes and REST API to support:

- create a user.
- retrieve town information by invitationID that embedded in invitation link.

In order to do invitation system, we added classes, REST api and listener to support:

- retrieve all in-system user (who created a username).
- Invite a user to a given town.
- retrieve a town's invitationID.
- one listener for each user that subscribes to invitation.

CRC cards

Class Name: ActiveUser	
States: ● **id:** the unique identifier of the user ● **username:** the username of the user ● **token:** the token used to subscribe the listener	
Responsibilities	**Collaborators**
Representing a active user in the system	CoveyUserController

Class Name: (interface) CoveyInvitationListener	
States: ● **id:** the id of this listener's owner	
Responsibilities	**Collaborators**
A listener that subscribed to invitation events. Once received invitation, it will notify the socket's owner the invitation information (coveyTownID and FriendlyName). Also, it subscribes to disconnect event.	CoveyUserController Socket

Class Name: CoveyTownController (changed)	
Added States: ● **invitationID:** the generated id for invitation (outside user)	
Responsibilities	**Collaborators**
Except for the original use, we added an invitationID to this class. User haven't launched the app may use the invitationID to identify which town the user is trying to join.	CoveyTownsStore UserInvitationRequestHandlers

Class Name: CoveyTownsStore (changed)	
No Added States	
Responsibilities	**Collaborators**
Except for the original use, we added a method to find townController by InvitationID	CoveyTownsStore UserInvitationRequestHandlers

Class Name: CoveyTownController (changed)	
Added States: ● **invitationID:** the generated id for invitation (outside user)	
Responsibilities	**Collaborators**
Except for the original use, we added an invitationID to this class. User haven't launched the app may use the invitationID to identify which town the user is trying to join.	CoveyTownsStore
	UserInvitationRequestHandlers

Class Name: CoveyUserController	
States: ● **instance** ● **users**: The active users in the system ● **listeners**: The invitation listeners subscribe the invitations	
Responsibilities	**Collaborators**
This class stores all existing user in the system (who have created a username). This class is managing the users, including list all users, find a specific user, create a user, delete a user. Also, CoveyUserController stores the in-system invitations listeners. It sends the invitations to the given user.	ActiveUser
	UserInvitationRequestHandlers
	CoveyInvitationListener
	Socket

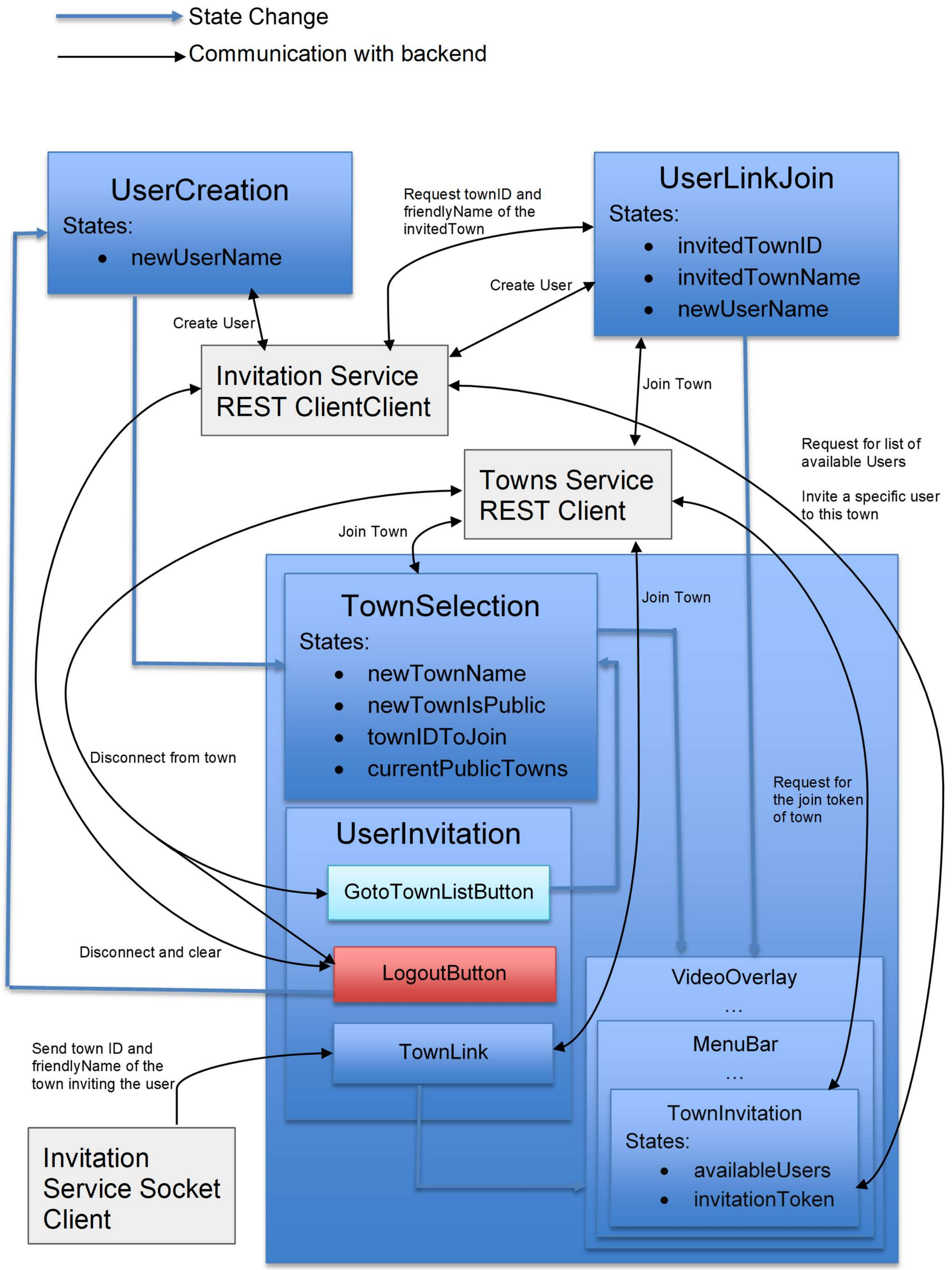
Class Name: UserInvitatoinRequestHandlers	
States: ● **CoveyUserController:** the controller stores the user and handles invitations. ● **CoveyTownsStore:** stores all the towns	
Responsibilities	**Collaborators**
This class exports functions that handles the requests related to users and invitations. Including following: create a user, subscribe the new user's socket to listen to listeners, list users, show the invitationID of a town, retrieve town information by invitationID, invite a existing user to a specific town.	CoveyTownController
	CoveyTownsStore
	CoveyUserController
	Socket
	Associating response/request interfaces

Frontend

We modified the frontend of Covey.Town so that it supports users and invitations.

We added services related to user, invitation and tried to keep all the original designs. We have designed to have a separated socket (user socket) to deal with invitation activities (i.e. receive invitation from other users) with the backend, aside from the original one dealing with town activities (town socket). Therefore, two sockets are created to capture different paths, where the town socket capturing */town*, and the user socket capturing */user*. The user socket is created after getting a valid response of creating a user from backend and is set to listening for 'invitedToTown' from backend.

Here's a graph showing the architecture of frontend components related to our project.



State Change
Communication with backend
UserCreation
States:
• newUserName
Create User
Request townID and friendlyName of the invitedTown
UserLinkJoin
States:
• invitedTownID
• invitedTownName
• newUserName
Create User
Invitation Service REST ClientClient
Join Town
Request for list of available Users
Invite a specific user to this town
Towns Service REST Client
Join Town
Join Town
TownSelection
States:
• newTownName
• newTownIsPublic
• townIDToJoin
• currentPublicTowns
Disconnect from town
Request for the join token of town
UserInvitation
GotoTownListButton
Disconnect and clear
LogoutButton
VideoOverlay
…
MenuBar
…
Send town ID and friendlyName of the town inviting the user
TownLink
TownInvitation
States:
• availableUsers
• invitationToken
Invitation Service Socket Client

React Components

UserCreation

This page has a text input box with some welcoming messages to let user to type in a username and a "Create" button. After hitting the "Create" button, the frontend will request the Invitation REST service to create a user and direct user to the TownSelection page.

UserInvitation

This component is a menu bar at the top of the page consisting of a "To Town List" button, a "Log out" button, and a drop-down list, the component "TownLink", lists all the names and IDs of towns that sent invitations to this user with a button to join the corresponding town and a button to delete the invitation. The "To Town List" button would disconnect user from the town and would be disabled if user is already in TownSelection page. The "Logout button" would disconnect user from backend totally, clear all states back to default and direct to the UserCreation page.

TownLink

This component is a drop-down list consisting of all the names and IDs of towns that sent invitations to this user with a button to join the corresponding town and a button to delete the invitation. This component would be refreshed when backend sends invitation messages to frontend through WebSocket with path */user*.

GoToTownListButton

This component is a button that would exit current town by disconnecting from the WebSocket with path /town and back to the TownSelection page upon clicking. This component would be disabled when IVideoContext.room is not connected since user would then either have already been on the TownSelection page or have not entered a username.

LogoutButton

This component is a button that would diconnect all connections, clear all states back to default, and back to the UserCreation page upon clicking. This component should be disabled exactly when IVideoContext.room is trying to, but have not yet, connect to Twilio.

UserLinkJoin

This page has similar layout and functionalities with the UserCreation, but it has a invitation token passed from the props that relates to a specific town, which was extracted from the address. Before first rendering, this component would request the Town REST service for the townID and friendlyName of the related town. After typing in a valid username in the text input box and click the "Create" button, this component would firstly send a request to Invitation REST service to create a user and then immediately request Town REST service to join the town.

TownInvitation

This component is placed as a button besides the TownSettings button at the bottom of the page in a town. Before first rendering, this component would request the Town REST service for the token of joining the current town. Upon clicking, it would pop up a window showing the invitation URL and a "Copy" button that would copy the invitation URL to user's clipboard. The window would also have a list of all available users requested from the backend, each entry consisting of their name, userID, and a "Invite" button that would send a request to backend to invite the corresponding user. This list would be refreshed every 2 seconds.

Reducer related

Func: loginController

Set up a socket capturing /user and listening for 'invitedToTown' from backend and update CoveyAppStates according to arguments. This function has similar functionality with GameController and should also be used similarly but with different arguments.

Service related

ServiceClient

Construct a Service API client. Specify a serviceURL for testing, or otherwise defaults to the URL at the environmental variable REACT_APP_ROOMS_SERVICE_URL. Deal with all communications to the backend through REST API. This class was originally "TownsServiceClient".