Will Passidomo

OfficeShare API(stage 2) Design Document
Renter API
10/30/14

Introduction

This document is to be used as the design for workers, to search for, and book available office space from independent, private providers as well as maintain an account and public profile. This service is in conjunctions with a Provider API which allows the respective office spaces to be listed and maintained.

Use Cases

The Renter API supports 2 different Use Cases.

1) Administrator updates static criteria lists (FacilityType, CommonFeatures, PrivateFeatures), Renters or OfficeSpace Providers are not allowed to add new features or types.
2) Renter
    a. Create a new Renter profile, updates its, or deletes it.
    b. searches (queries) office spaces using search criteria
    c. places reservations for Office Spaces and handles payment
    d. leaves Ratings for OfficeSpaces and Providers

Overview

Like with the Provider API, the problem that this program aims to solve is that of inefficiency. In this case, it is the inefficiency month to month, yearly or generally overprices leasing structures. In the case of a freelance worker, regular, predictable workload and hours mean that, in order to be productive, it is not beinifical it lock into a fixed term and price contract with a traditional office space lease, that is to say if the option for leasing a work space for a single person is an option. In the case of a small startup team, the workload demands dedicated space, accessible by members, but the unpredictable nature of the work means that signing a lease past a month or two, for underfunded companies, could mean that a fizzled launch could translate to long lasting personal liabilities which could haunt the lease-signer long after the idea has fizzled.

Enter the Renter API of Squaredesk. The Renter service allows for the parties discussed above, amongst others, to take advantage of a more flexible solution to their needs. Squaredesk Renter API allows Renters to search for work spaces sized and priced for their individual or small business needs. The OfficeSpace on the Renter API are populated by private citizens offering up unused space in their own homes, garages, barns and more. This means that Renters will not be dealing with a Landlord who has a fixed bottom line or an industry set up to trap the little guy, but rather an individual who has no more motivation than to just use their extra space

and make a little extra cash. Rates and rental period are low and manageable and offers a solution whether its for your small business, or the small business is yourself.

Requirements

1) to support creation, update and deletion of Renters. Renter is an optional profile of a User (may have Renter rofile, Provider profile or both). Renters have a Nam, and Contact, an Image, and a PaypalAccount. All of these must be updateable, creatable and deletable (as long as profile still meets minimum criteria of 1 of each).
2) To support the submittal of Ratings by Renters. Renters may submit ratings about OfficeSpaces or OfficeSpaceProviders. Ratings include a star count (1-5) and comment and a time stamp, as well as reference to the rater and the rated.
3) To support the querying of available office spaces by the following criteria. Private Features, Common Features, Facility Type, Average Rating, Location, and to be able to check availability for a optionally provided start date and end date.
4) To support the booking of office spaces through reference received from query results. Booked office spaces must not be able to be returned in query results specifying the booked time and must not be able to be double booked. Part of this functionality is a method isAvailable() which test for availability for a given Office Space on a given date or range of dates.

Class Dictionary

RenterAPI

The RenterAPI class contains all of the client-side public methods for the implementation. RenterAPI should rely exclusively on returning and accepting only DTO objects, but, due to a different implementation style during the ProviderAPI implementation, the RenterAPI must support legacy, non-DTO objects implemented in the first ProviderAPI.

This class is used for fulfilling the requirements listed in the Requirement section and allows an encapsulation of the methods from the client. The RenterAPI mearly redirects method calls to the singleton RenterServiceImplementation class, a package-private class which is responsible for directly interacting with the implementation.

The RenterAPI class is also, although non is implemented in this design, responsible for security and the approval and distribution of authTokens

Method Name	Signature	Description
updateRenter	(authToken:String, renter:RenterDTO):RenterDTO	Allows a Renter to update their Name,

		Contact, PayPal, and Image by modifying their corresponding DTO's and submitting a RenterDTO. Returns the updated RenterDTO
deleteRenter	(authToken:String, renterID:UUID):void	Allows a renter to delete themselves
getRenter	(authToken:String, renterID:UUID):RenterDTO	Returns a RenterDTO which corresponds to a RenterObject with the specified UUID
submitQuery	(authToken:String, query : QueryDTO, renterID:UUID) : List<OfficeSpace>	Submits a query with the criteria specified in the DTO and returns a list of matching OfficeSpaces
makeReservation	(authToken:String, renterID :UUID, reservation :ReservationDTO)	Checks to make sure officespace is available. Creates a ReservationObject and returns the corresponding ReservationDTO
createRenter	(authToken:String, renter:RenterDTO, userID:UUID):RenterDTO	Creates a renter with the corresponding RenterDTO object and associated it wil the the corresponding User account. Returns a renterDTO based on the new Renter instance
submitRating	(authToken:String, rating:RatingDTO):RatingDTO	Submits a rating to the system based on the contents of the RatingDTO
removeRating	(authToken:String, ratingID:UUID	May only be applied to ratings which the renter themselves left
getRatings	(authToken:String, renterID:UUID):List<Rating>	Returns a list of Rating objects for the renter with Renter
getFacilityTypes	(authToken:String) :List<FacilityType>	Returns a list of all available FacilityTypes
getCommonFeatures	(authToken:String)	Returns a list of all

	:List<CommonFeature>	available CommonFeatures
getPrivateFeatures	(authToken:String) :List<PrivateFeatures>	Returns a list of all available PrivateFeatures

Renter

The Renter class is a "profile" associated with a User. A user may have a renter profile from which they may search for and reserve OfficeSpaces and rate OfficeSpaces and OfficeSpaceProviders, renters may not list officeSpaces as an OfficeSpaceProvider may.

Method Name	Signature	Description
reserveOfficeSpace	(officeSpace:OfficeSpace, startDate:Date, endDate:Date, rentalRate:UUID):ReservationObject	Creates and returns a ReservationObject if reservation is made or Null if it was unable to be made. Method checks whether rentalRate:UUID is a valid offering for OfficeSpace before booking and checks Scheduler for date availability
submitOfficeSpaceRating	(officeSpace:OfficeSpace, rating:Rating, reservation:ReservationObject):void	User may only review office only which they have used, method verifies this by requiring that the reservation object passed must be for the space or provider being rated and by the renter who is rating

Association Name	Type	Description
name	Name	For purposes of identifier Renter by their full legal

		name
contact	Contact	Contact information for renter
image	Imagec	Profile picture for the renter
paypalAccount	PayPalAccount	PayPallAccount associated for the purpose of payment
ratings	List<Rating>	Ratings for this renter (provided by Office Space Providers)

Renter DTO

The RenterDTO class is a Data Transfer Object representing the Renter class. It contains 4 associations with NameDTO, ContactDTO, ImageDTO, and PayPalDTO and only setter and getter methods

Association Name	Type	Description
nameDTO	NameDTO	For purposes of identifier Renter by their full legal name
contactDTO	ContactDTO	Contact information for renter
imageDTO	ImagecDTO	Profile picture for the renter
paypalDTO	PayPalDTO	PayPallAccount associated for the purpose of payment

QueryBuilder

The QueryBuilder class is a class which handles searches for available office spaces for the following criteria, common features, private features, location, minimum rating, facility type, start date and end date. The QueryBuilder is backed by the KnowledgeEngine.KnowledgeGraph class which is responsible for indexing searchable Office Spaces and handles queries in the form of triples. It is the responsibility of the QueryBuilder class to gather the search criteria from the Client, transform the criteria into triples, readable by KnowledgeEngine.KnowledgeGraph, iteratively execute triple queries, and translate and form result sets of Office spaces, readable by the client.

Clients may add “conditions” to the search using the overloaded addCriteria() method, and call execute() on the object when search criteria is completely loaded. Execute() method will return OfficeSpace objects that meet ALL minimum criteria listed in the conditions. Support for “or” searches may be added at a later date, in

the mean time it is suggested that client should perform multiple queries and combine results on their own.

Method Name	Signature	Description
addCondition	(commonFeatures:List<CommonFeature>):void (privateFeatures:List<PrivateFeature>):void (coordinate:Coordinate):void (address:Address):void (minimumRating:int):void (facilityType:FacilityType):void	Gives the ability to "build" a query. When execute() is called, only results which satisfy all condtions passed in addCondition() will be returned
setSartDate()	(startDate:Date):void	Add a desired start dates to the query
setEndDate()	(endDate:Date):void	Add a desired end date to the query
executeQuery()	():List<OfficeSpaces>	Calls toTriple() and KnowledgeEngine.KnowledgeGraph.executeQuery iteratively after condtions have been fully added to query, then runs results against Scheduler.isavailable() to check for available dates
updateIndexes	(officeSpace:OfficeSpace):void	A static method which turns an office space and its features into a series of triples and updates indexes in KnowledgeGraph by calling importTriples(tripleList:List<Triple>)
toTriple	(condition:String):Triple	A static method which uses object's toString method to provide textual representation of condition and gives it the proper predicate and "?" as subject in order to form a ready triple
getOfficeSpace	(Triples:List<Triple>):List<OfficeSpace>	A static method which iteratively calls KnowledgeEngine.KnowledgeGraph.executeQuery and translates results into corresponding OfficeSpace instances

QueryDTO

The QueryDTO class is a Data Transfer Object for a Query. QueryDTO contains associates with querable (Implements Searchable) critera. Although it would be simpler to have addCondition(Searchable) as a uniform method for adding criteria, I designed this class as having an overloaded addCondition() method with each of the separate Searchable objects as criteria for the sake of clarity on the client side.

Method Name	Signature	Description
addCondition	(commonFeatures:List<CommonFeature>):void (privateFeatures:List<PrivateFeature>):void (coordinate:Coordinate):void (minimumRating:int):void (facilityType:FacilityType):void	Gives the ability to "build" a query. When execute() is called, only results which satisfy all condtions passed in addCondition() will be returned

Association Name	Type	Description
commonFeatures	CommonFeature	Commonfeatures for results to be filtered by
Private features	PrivateFeatures	Privatefeatures for results to be filtered by
Coordinate	Coordinate	Coordinate for results to be filtered by
minimumRating	Int	Minimum ratings for results to be filtered by
facilityType	FacilityType	facilityType for results to be filtered by

ReservationObject

The ReservationObject call provides a record or marker of a certain officeSpace being reserved from a start date to an end date for a certain legal rate be a Renter. The ReservationObject is in charge of tracking the payment status and the outstanding balance the reservation. All fields in the ReservationObject aside from balance and paymentStatus are immutable

Method	Signature	Description
makePayment	(payPalAccount:PayPalAccount,amount:d	For making partial

	ouble):boolean	payments, i.e a deposit on a space or under the terms of a payment plan. Returns “true” if payment is accepted, “false” if payement is rejected
makePayment	(payPalAccount:PayPalAccount):boolean	For making full payments. Returns “true” if payment is accepted, returns “false” if payment is rejected

Association	Type	Description
startDate	Date	The date the reservation is to start
endDate	Date	The date the reservation is to end
officeSpace	OfficeSpace	The officeSpace to be rented
Renter	Renter	The renter to rent the officeSapce

ReservationDTO

The ReservationDTO is a Data Transfer Object for a reservation Object. It contains fields related to the corresponding ReservationObject and only has getter and setters

Association	Type	Description
startDate	Date	The date the reservation is to start
endDate	Date	The date the reservation is to end
officeSpaceID	UUID	The officeSpace to be rented
RenterID	UUID	The renter to rent the officeSapce

Scheduler

The Scheduler class is a singleton class used to mange the OfficeSpace Object’s available time’s and ReservationObjects. In a future implementation, Office Spaces will have certain times when they are and when they are not available.

Reservations are made through the Scheduler and the Scheduler is responsible for creating and destroying ALL ReservationObjects. The Scheduler is also responsible for tracking and providing Lists of reservations for OfficeSpaces and OfficeSpaceProviders.

Method Name	Signature	Description
makeReservation	(officeSpace:OfficeSpace, renter:Renter, startDate:Date, endDate:Date, rate: Rate) : ReservationObject	Reserves a OfficeSpace for renter from startdate to enddate at rate. Returns ReservationObject if able to make reservation, returns Null if unable to make reservation..throws ImproperRateError and ScheduleConflictError
cancelReservation	(reservation:ReservationObject):boolean	Cancels an existing reservation, throws ReservationDoesNotExistError if reservation does not exist in the system and CannotCancelPastStartDateError if client attempts to cancel reservation after the reservation has already started.
getReservations	(officeSpace:OfficeSpace):List< ReservationObject>	Returns a List of reservations, past and future for the provided officeSpace
getReservations	(officeSpaceProvider:OfficeSpaceProvider):List<ReservationObject>	Returns a List of reservations, past and future for the provided OfficeSpaceProvider
IsAvailable	(officeSpace:OfficeSpace, startDate:Date, endDate:Date):Boolean	A method to check the availability of an office space, used primarily to filter results from QueryEngine by date
getInstance	():Scheduler	A static method which returns the singleton instance of the Scheduler class

Rateable

Rater

Searchable

Implementation Details

The Renter API was implemented using Data Transfer Objects and a single class for all public API methods, the RenterAPI class. The purpose for implementing the RenterAPI class and the DTO classes was two fold, 1) it prepared the system for eventually acting as a distributed system and 2) it easily allowed me to encapsulate the implementation and separate it from the interface. While #1 is less important, as it is not a current requirement, the encapsulation of the implementation from the interface was significant in that it allows me to change the implementation at will without having to worry about damaging the Interface or breaking and client programs running on the Interface. This is perfect for this sprint, because I feel like there is a lot of optimization that could be done to the implementation code, but now I have to option of continuing forward with a working interface and get back to the optimization in the future when there is more time and not have to worry about the interface changing.

The code implemented here made good use of several custom Interface classes. The Rateable interface was implemented by Renter OfficeSpace, and OfficeSpaceProvider, Rater was implemented by OfficeSpaceProvider and Renter and Searchable was implemented by CommonFeature, PrivateFeature, Coordinate, AverageRating, and FacilityType. These interfaces allowed for what would have otherwise been complex operations and validations to be performed in a few lines of code. These Interfaces also make the code much more extensible as other features may be able to be made searchable very easily, if the decision is made later to increase searchable criteria.

As of now, the implementation of Query and search functionality is limited to returning results which meet ALL criteria listed. In the future, it would be good for renters to be able to search for spaces which meet either all of a combination of criteria or one of several different criteria. In order to add more flexibility to the system. I chose to ignore this feature this time through the RenterAPI as the lack of offering does not diminish functionality since the client has the option of just executing search criteria separately then joining the results if they wish to give their users this functionality.

Testing

I wrote a testing program which acts in two phases, first, it loads a number of Users, OfficeSpaceProviders, OfficeSpace, FacilityTypes, CommonFeatures and PrivateFeatures into the system to act as “seed” elements, since they cannot be loaded through the RenterAPI. I then wrote a test Program through the RenterAPI which did the following steps:

1) created Renter1;

2) created Renter2
3) created Renter3
4) retrieved List of all Renters;
5) deleted Renter3
6) update Name, Contact, PayPal, and Image information of renter2;
7) retrieved Renter1 based on UUID (of created Renter)
8) retrieved a list of all facilityTypes;
9) retrieved a list of all privateFeatures;
10) retrieved a list of all commonFeatures;
11) Renter1 performs a blank query to return all office spaces
12) Renter2 performs a query based on specific criteria
13) Renter 1 makes a reservation for a space;
14) Renter2 makes another reservation for a space;
15) Renter2 cancels his reservation;

Risks

There are numerous risks facing this implementation of RenterAPI, many of which are due to gaps in the requirements and some are due to the implementation.

Risks stemming from the requirements are as follows:

1) Memory- there needs to be a database used for storage, the local memory will not fare well as more users apply to the service.
2) Persistance- there is no persistence in the program, input will be lost if the program stops running
3) Security- a full implementation leveraging authToken's need to be implemented to prevent malicious users.
4) Distribution- the program risks stunted growth unless it can be changed to run on a distributed network

Risk which stem from this implementation are as follows;

1) incomplete data validation, error controls- there is not enough checking implemented in this program as it is in its current state, Im not sure exactly what I missed because I am rushing to turn it in, but there are definitely gaps in areas, such as checking if one person has multiple reservations at the same time or checking if an office space owner is renting his own space that should be implemented
2) lack of optimization- without further optimization to the program, I doubt it could perform well enough on a distributed network. The most previlant issue I can think of is the implementations choice of data structures, too many ArrayLists where there should be set's;