CMPT 756.211 Team-A Term Project Submission

127.0.0.1

Find your home

Section 1 - Problem Statement

Problem Domain

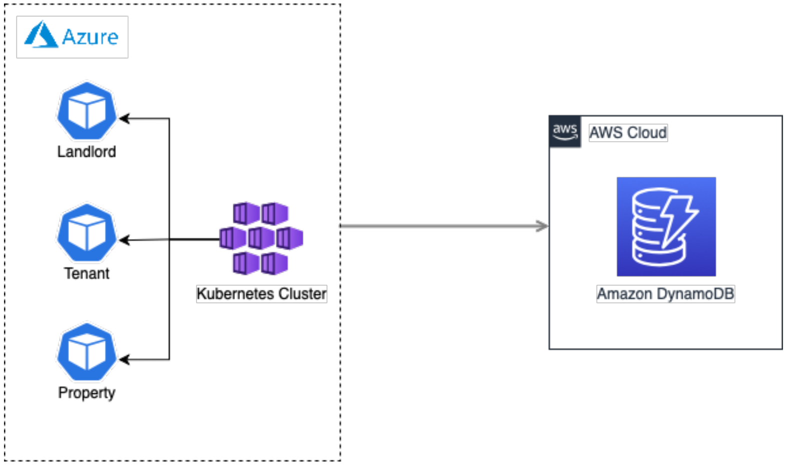
Finding an ideal house to stay in is nothing short of a pipe-dream sometimes. Landlords are constantly looking at dozens of tenant applications, without an easy way of choosing the perfect tenant. Landlord-tenant relations tend to go haywire if the landlord's requirements are not met by the tenant or vice-versa. Landlords don’t have access to tools and resources that allow them to effectively manage their rental properties. **This causes chaos** — both for landlords and their tenants. SINs are exchanged without much thought, rent cheques show up late in the mail, leases aren’t legally binding, and the list goes on and on. Consequently, there is a torrent of landlord-tenant problems that affect both parties and we need a solution to streamline these pressing problems.

127.0.0.1 - Find Your Home is an online platform that enables landlords and their tenants to gain a common consensus and enhance transparency. A landlord would want to list and publicize his properties, manage his tenants and attend to service requests to ensure meeting his side of the deal. On the other hand, a potential tenant would want to find and compare properties or raise service requests in their existing property.

Our solution aims to simplify the interactions amongst tenants, landlords and their properties by bringing them into a common domain and establishing a relationship that alleviates this plethora of problems making the pipe-dream a reality.

Application Architecture

Our application consists of three entities - Tenant, Landlord and Property. Each of these entities has significant interaction amongst themselves.



The services for the following entities are running on a Kubernetes cluster hosted on Azure and the database used is DynamoDB, a NoSQL DB running on AWS Cloud. Operations available for each of the entity are listed below -

Tenant

- Create Account - Create an account as a tenant
- Update Account - Update the tenant account details
- Delete Account - Delete an existing tenant account
- Login - Login to the tenant account
- Logout - Logout from the account

Landlord

- Create Account - Create an account as a landlord
- Update Account - Update the landlord account details
- Delete Account - Delete existing landlord account
- Login - Login to the landlord account
- Logout - Logout from the landlord account
- Create Property - Add a property listing
- Delete Property - Remove a property listing

Property

- List Properties - Get property listings based on the location
- View User Property - Get a list of properties owned by the landlord or rented by the tenant
- View Property Details - View details of a particular property listing
- Create Service Request - Create a service request as a tenant to raise complaints
- View Service Request status - View the status of the service request
- Resolve Service Request - Resolve the service request as a landlord

Specification of REST API (Microservices Contract)

Version: v1
Service: Gateway
Visibility: Public
Domain: Ingress-gateway
Serialized Data/Content-Type: json/xml

API	**Description**
/api/v1/landlord	Landlord service URL
/api/v1/tenant	Tenant service URL
/api/v1/property	Property service URL

Version: v1
Service: **Landlord**
Visibility: Public
Domain: Landlord user
Serialized Data/Content-Type: json/xml

API	**Description**	**Request Body**	**Response Body**	**HTTP Response Code**	**Error Code**	**Request Example**	**Response Example**
POST - /api/v1/landlord/	CREATE landlord account	**Body: { email:** string, **fname:** string, **lname:** string, **username:** string, **password:**string **}**	{ “user_id”: “L_”+username }	200	500	POST https://host:5000/api/v1/landlord/	{ user_id: “L1234”}

UPDATE - /api/v1/landlord/	UPDATE landlord account	**Body: {** <key>: <new_value> }	OK response	200	500	POST https://host:5000/api/v1/landlord/<user_id>	{ Message: ok }
DELETE - /api/v1/landlord/	DELETE landlord account	**Body: { token**: SessionToken}	JSON of response from aws	200	500	DELETE https://host:5000/api/v1/landlord/	{ Message: ok }
PUT - /api/v1/landlord/login	Login into account	**Body:{ user_id**: “L_”+username, **password**:password}	A session token that would be used to validate user’s session {“UserContext”: SessionToken}	200	500	PUT https://host:5000/api/v1/landlord/login	{“UserContext”: qwe123qwedfs9878678sd }

PUT - /api/v1/landlord/logout	Logout from account	**Body: { token**: SessionToken}	Confirmation Message	200	500	PUT https://host:5000/api/v1/landlord/logout	{ Message: ok }
POST - /api/v1/landlord/property	CREATE landlord property	**Body: { name:** string, **address:** string, **availability:** string, **beds:** string, **baths:**string , **rent:** string, **facilities:** list, **user_id:** “L_”+username **}**	{ “property_id” : string }	200	500	POST https://host:5000/api/v1/landlord/property	{ property_id: “P_123”}

DELETE - /api/v1/landlord/property	DELETE landlord property	**Body: { property_id**: string}	Confirmation message	200	500	DELETE https://host:5000/api/v1/landlord/property	{ Message: ok }

Version: v1
Service: **Tenant**
Visibility: Public
Domain: Tenant user
Serialized Data/Content-Type: json/xml

API	**Description**	**Request Body**	**Response Body**	**HTTP Response Code**	**Error Code**	**Request Example**	**Response Example**

POST - /api/v1/tenant/	CREATE tenant account	**Body: {** **email:** string, **fname:** string, **lname:** string, **username:** string, **password:**string **}**	{ “user_id”: “T_”+username }	200	500	POST https://host:5000/api/v1/tenant/	{ user_id: “T_sam123”}
UPDATE - /api/v1/tenant/	UPDATE tenant account	**Body: {** <key>: <new_value> }	OK response	200	500	POST https://host:5000/api/v1/tenant/<user_id>	{ Message: ok }
DELETE - /api/v1/tenant/	DELETE tenant account	**Body: { token**: SessionToken}	JSON of response from aws	200	500	DELETE https://host:5000/api/v1/tenant/	{ Message: ok }

PUT - /api/v1/tenant/login	Login into account	**Body:{ user_id**: “T_”+username, **password**:password}	A session token that would be used to validate user’s session	200	500	PUT https://host:5000/api/v1/tenant/login	{“UserContext”: 123lkj312hlkaluw09789kn }
PUT - /api/v1/tenant/logout	Logout from account	**Body: { token**: SessionToken}	Confirmation Message	200	500	PUT https://host:5000/api/v1/tenant/logout	{ Message: ok }

Version: v1
Service: **Property**
Visibility: Public
Domain: Properties
Serialized Data/Content-Type: json/xml

API	Description	Request Body	Response Body	HTTP Response Code	Error Code	Request Example	Response Example
GET - /api/v1/property/city	View all property ids for a location		JSON of Property ids {Properties:<list_of_property_ids> }	200	500	GET https://host:5000/api/v1/property/city/<city_name>	{Properties: { “P_123”, “P_876”, “P_634” } }
GET - /api/v1/property/user	View all user property ids		JSON of Property ids {Properties:<list_of_property_ids> }	200	500	GET https://host:5000/api/v1/property/user/<user_id>	{Properties: { “P_193”, “P_976”, “P_134” } }

GET - /api/v1/property	View property details		JSON of Property details {Property_id:**{** **name:** string, **address:** string, **availability:** string, **beds:** string, **baths:**string , **rent:** string, **facilities:** list, **user_id:** “L_”+username **}**	200	500	GET https://host:5000/api/v1/property/<property_id>	**{Property_id:{** **name:** P_6534, **address:** Burnaby, **availability:** 2, **beds:** 3, **baths:** 2 , **rent:** 2500, **facilities {** wifi,heating,electricity,hydro,gym **},** **user_id:** “L_john738” **}**
PUT - /api/v1/property/service_req	CREATE service request for property	**Body: {** **user_id:** “T_”+username, **property_id:** string,	*Service Request id*	200	500	PUT https://host:5000/api/v1/property/service_req	{request_id: 632987 }

		request: string**}**	{request_id: int }				
GET - /api/v1/property/service_req	View service request status		JSON of Service request details of a property {**Property_id**:**{** **t_username**:{ **request_id:{** **query:** string, **resolved:** bool, **}}}**	200	500	GET https://host:5000/api/v1/property/service_req/<property_id>	**{Property_id**:**{** t_arun87:**{request_id:{** **query:** “My thermostat is not working”, **resolved:** False, **}}}**
UPDATE - /api/v1/property/service_req	UPDATE service request status	**Body: {** **property_id:** string, **user_id:** “T_”+username,	OK response	200	500	POST https://host:5000/api/v1/property/service_req	{ Message: Request resolved }

		request_id: string, }					

Database Schema (DynamoDB)

Table: user_details-teamA756

Tag	**Value**	**Comment**
u_id	string	The tenant id is of the format **t_<username>** and the landlord id is **l_<username> (KEY)**
username	string	Username for either the tenant or landlord
password	string	Password combination to log into the tenant/landlord account
fname	string	First name of tenant/landlord
lname	string	Last name of the tenant/landlord
email	string	The email address of the tenant/landlord
contact	string	Unique id not interpreted by DynamoDB
properties	list	Properties owned by landlord stored as a list of property_ids or p_ids

Table: property-teamA756

Tag	**Value**	**Comment**
p_id	string	Property ID **(KEY)**
address	string	Property address
availability	bool	Flag to check if the property is available for renting or occupied
rent	int	Expected rent
beds	int	Number of bedrooms
baths	string	Number of bathrooms
facilities	string	Facilities and amenities included
u_id	string	Landlord u_id (<l_username>)

Table: service_requests-teamA756

Tag	**Value**	**Comment**
p_id	string	Property ID **(KEY)**
u_id	string	Tenant ID who raised the request (<t_username>)

q_id	string	Request ID
query	string	Message for service request
resolved	bool	Flag to check if the request is resolved or not

Section 2 - Github Repo Guide

This section explains the project file structure

Path	**Note**
/readme.md	Project overview and usage instructions
/wiki	Documentation for the project
/wiki/report	Project report
/wiki/video	Project demo video
/code/service-landlord/	Code for landlord API
/code/service-tenant/	Code for tenant API
/code/service-property/	Code for property API
/code/db/	Code for DynamoDB API service
/IaC	Configuration for environment setup
/IaC/cluster/	Kubernetes manifests, Cloudformation templates and user configuration
/IaC/gatling/	Gatling simulation files and sample csv’s

/IaC/tools/	Shell scripts for configuration
/IaC/logs/	Application logs

Section 3 - Reflection on Development

What did you observe from applying and using the scrum methodology? What worked well? What didn’t? What surprised you?

We followed the following elements of the scrum methodology:

- Held a sprint planning meeting at the start of the project
- Held biweekly standups for the duration of our sprint
- Started working on a product backlog and defined milestones to implement items
- Created Kanban planning board with issues spread across different phases of development
- Classify estimated project development into different sprints based on milestones

What went well:

- The Scrum methodology allowed us to effectively focus our efforts and optimize project development
- Features were prioritized based on importance and high-priority features were picked first
- Scrum calls facilitated the quick resolution of issues and easy decision-making
- Transparency was achieved by using GitHub Issues and project board features for task tracking and distribution which enabled effective coordination amongst the team
- Artifacts were inspected regularly and adaptations were made to increase productivity and mitigate risks
- Our team was cross-functional, self-managing, allowed each member to work independently and upheld Scrum values

What didn’t go well:

- Due to project members working in different timezones, we faced problems in scheduling standup calls

- Some members faced technical issues which were managed appropriately

Reflect on the readings over the course of the term. What ideas were you able to apply? How did these turn out?
The readings on 'Scrum artifacts' provided a thorough understanding of the Scrum elements and the modification required for our project. The reading on 'Microservices' provided significant insights in developing the project architecture.

If you have professional experience with Scrum, how did your team perform in comparison to past teams?
Our team was committed, transparent, respectful, cross-functional, and collaborative in project planning.

Section 4 - Analysis

Section 4.1 - Coverage Simulation

We came up with the following plan for our coverage simulation. For each of the endpoints, our test suite would run three types of tests.

- **Basic Positive Tests -** Executing API calls with **valid required parameters**.
- **Negative Testing (valid input) -** Executing API calls with **valid input** that attempts **illegal operations**.
- **Negative Testing (invalid input) -** Executing API calls with **invalid input**.

The complete plan for the coverage simulation is explained below in the tables.

The coverage simulation exercise allowed us to validate the functional working of our API. We tested a variety of scenarios to ensure that the flow of the application does not break. The coverage simulation ran in parallel with creation of APIs. Each team member co-wrote the simulation scripts as soon as the API was implemented. This ensured the correctness of the newly created API and also checked for any breaks in the existing application. We plan to further include more test cases such as negative testing with valid and invalid inputs to further test the robustness of our application.

Endpoint: /api/v1/landlord		
API Call	**API Description**	**Test Action Description**
Basic positive tests		
POST /landlord/	Create Landlord Account	Landlord can create a new account.
PUT /landlord/<user_id>	Update Landlord Account	Landlord can update his account
	Delete Landlord Account	A landlord is able to delete their own account
POST /landlord/property	Create Property	Landlord can create a property
	Delete Property	A landlord is able to delete their own property
PUT /landlord/login	Login	A landlord should be able to login with their username and password
PUT /landlord/logoff/<user_id>	Logout	Landlord successfully logs out by providing their username
Negative testing – valid input		

Negative testing – invalid input		

Endpoint: /api/v1/property		
API Call	API Description	Test Action Description
Basic positive tests		
	List Properties for a Location	Tenant can view properties based on location
	View User Properties	Tenant/Landlord get a list of their properties
	View Property Details	The user is able to view the detailed description of the property
POST/property/service_req	Create Service Request	A tenant can create multiple service requests for the same property. Multiple tenants can create different service requests for the same property
	View Service Request status	Tenant can check the status of their service request
PUT /property/service_req_update/<query_	Update Service Request	A tenant can update their existing service request by passing the query

id>		id
PUT /property/resolve_req/<query_id>	Resolve Service Requests	Landlord can resolve a service request raised for their property by passing relevant details
Negative testing – valid input		
Negative testing – invalid input		

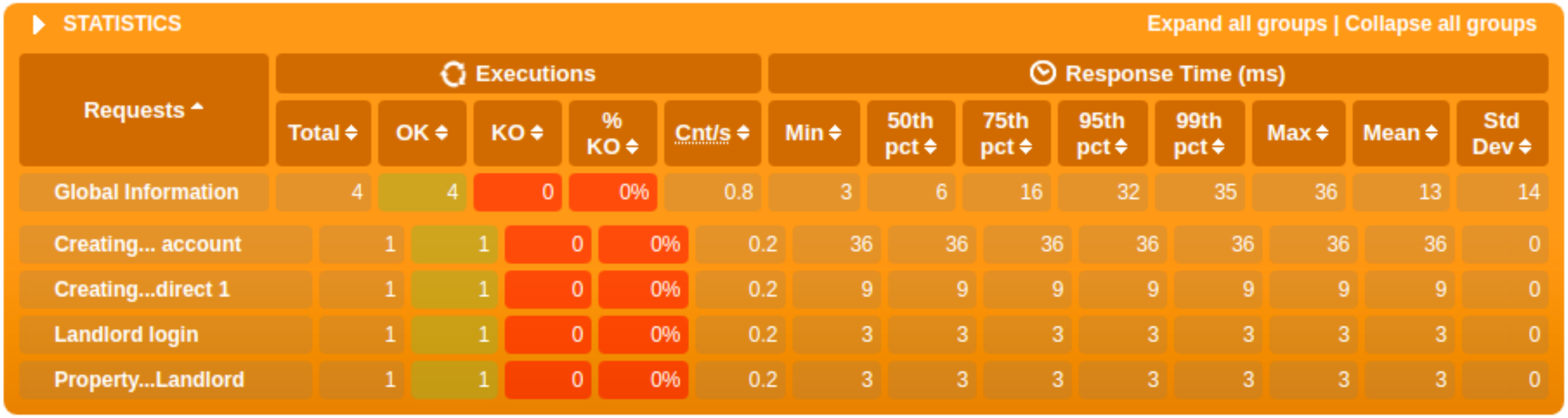
Endpoint: /api/v1/tenant		
API Call	**API Description**	**Test Action Description**
Basic positive tests		
POST /tenant/	Create Tenant Account	A tenant is able to create an account that has user_id generated as T_<username>. The details are updated in the user_details table.
PUT /tenant/<user_id>	Update Tenant Account	A tenant is able to update their account and change their details.
DELETE /tenant/<user_id>	Delete Tenant Account	A tenant should be able to delete their account from the application.

PUT /tenant/login	Login	Tenant should be able to login into the tenant account and generating a authorization token
PUT /tenant/logoff	Logout	Able to logout from the tenant account.
Negative testing – valid input		
Negative testing – invalid input		

Section 4.1.1 Coverage Results

Section 4.1.1.1 Landlord Service

Gatling simulation was successfully run by hitting all the APIs created until Milestone-2 Sprint Backlog. The following are the results of our Positive Testing Simulation for the Landlord Service.

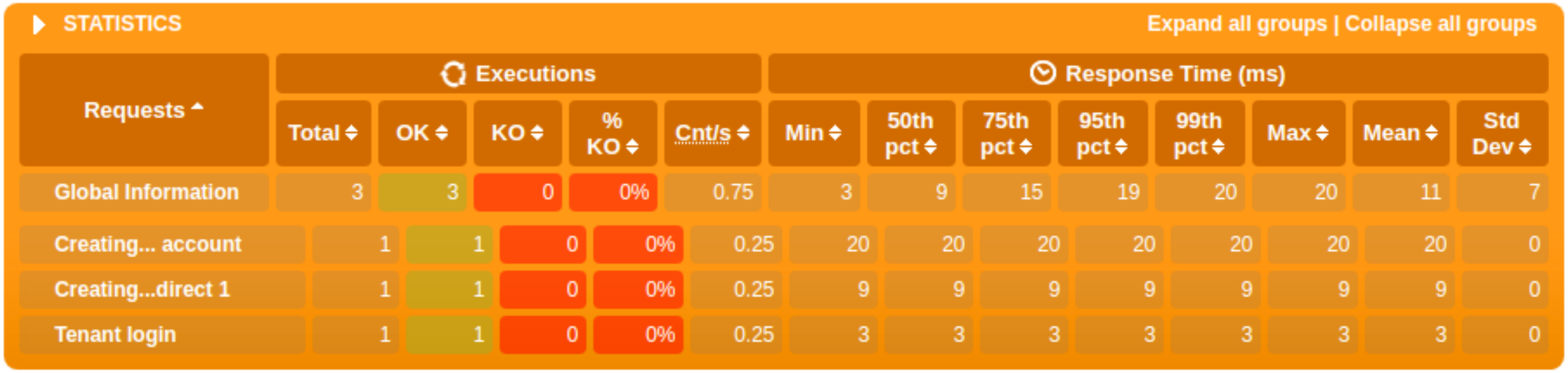


Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	4	4	0	0%	0.8	3	6	16	32	35	36	13	14
Creating... account	1	1	0	0%	0.2	36	36	36	36	36	36	36	0
Creating...direct 1	1	1	0	0%	0.2	9	9	9	9	9	9	9	0
Landlord login	1	1	0	0%	0.2	3	3	3	3	3	3	3	0
Property...Landlord	1	1	0	0%	0.2	3	3	3	3	3	3	3	0

Section 4.1.1.2 Property Service

Section 4.1.1.3 Tenant Service

Gatling simulation was successfully run by hitting all the APIs created until Milestone-2 Sprint Backlog. The following are the results of our Positive Testing Simulation for the Tenant Service.



Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	3	3	0	0%	0.75	3	9	15	19	20	20	11	7
Creating... account	1	1	0	0%	0.25	20	20	20	20	20	20	20	0
Creating...direct 1	1	1	0	0%	0.25	9	9	9	9	9	9	9	0
Tenant login	1	1	0	0%	0.25	3	3	3	3	3	3	3	0

Section 4.2 - Load Testing Simulation

<How did the load simulation help with testing of your completed system? What types of failures did you simulate and what were the outcomes? How did your application respond to various disturbances to the network?>

Our load testing plan aims to cover these read and write requests for the API calls to measure the system's responsiveness, throughput and robustness. In our plan, we will be ramping up the number of users upto the point where our services start showing errors. This should give us a fair idea of which services act as a bottleneck in our entire application. This would also be helpful in deciding the appropriate resolutions for the problems found.

API	Request Count	Min Response Time	Max Response Time	50th percentile	75th percentile	95th percentile	99th percentil e	%error
POST /landlord/								
POST /landlord/property								

PUT /landlord/login								
PUT /landlord/logoff								
POST/property/service_req								
POST /property/service_req_update/<query_id>								
POST /property/resolve_req/<query_id>								
POST /tenant/								
PUT /tenant/<user_id>								
DELETE /tenant/<user_id>								
PUT /tenant/login								
PUT /tenant/logoff								