CMPT 756.211 Team-A Term Project Submission

127.0.0.1

Find your home

Video URL - https://youtu.be/ATlthWBY0pA

Team	Team-A				
Section	G100				
Team Member Name	Inderpartap Cheema	Rishabh Jain	Navaneeth M	Anuj Saboo	Tavleen Sahota
Email (SFU)	ischeema@sfu.ca	rja40@sfu.ca	nmm14@sfu.ca	asaboo@sfu.ca	tss13@sfu.ca
Email (G-Suite/Gmail)	inderpartap15@gmail.com	rishabhjain2017@gmail.com	navs1993@gmail.com	saboo.anuj17@gmail.com	tavleenssahota@gmail.com
Github ID	inderpartap	rja40	jediXNavi	anuj-saboo	tssahota
Additional Notes					

Section 1 - Problem Statement

Problem Domain

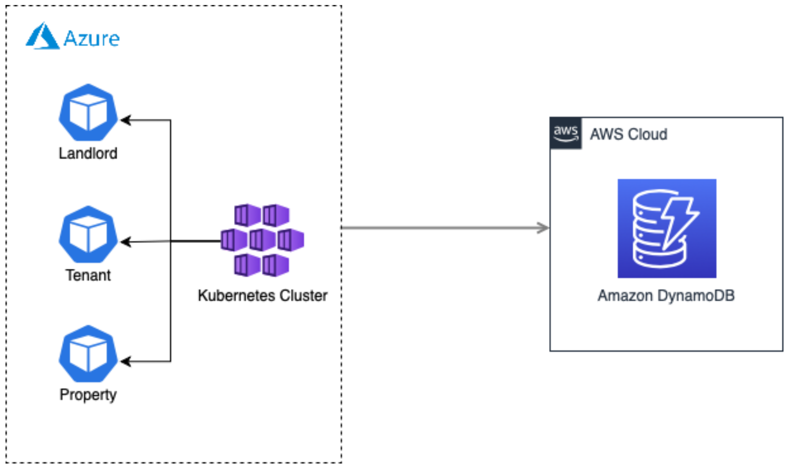
Finding an ideal house to stay in is nothing short of a pipe-dream sometimes. Landlords are constantly looking at dozens of tenant applications, without an easy way of choosing the perfect tenant. Landlord-tenant relations tend to go haywire if the landlord's requirements are not met by the tenant or 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

- Create Service Request - Tenant creates a new service request for his property
- Update Service Request - Tenant can update his/her service request
- Tenant Property Application - Tenant applies for property

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
- Resolve Service Requests - Landlord resolves service request

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
- View Service Request - View all service requests created for landlord’s properties

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,**contact**: int **}**	{ “user_id”: “L_”+username }	200	500	POST https://52.139.9.83:80/api/v1/landlord/	{ user_id: “L_mike”}
UPDATE - /api/v1/landlord/<user_id>	UPDATE landlord account	**Body: {** **email:** string, **fname:** string, **lname:** string,	OK response	200	500	POST https://52.139.9.83:80/api/v1/landlord/L_mike	{ Message: Account Updated }

		password:string, contact: int }					
DELETE - /api/v1/landlord/<user_id>	DELETE landlord account		JSON of response from aws	200	500	DELETE https://52.139.9.83:80/api/v1/landlord/L_mike	{ Message: User Deleted }
PUT - /api/v1/landlord/login	Log 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://52.139.9.83:80/api/v1/landlord/login	{"jwt": qwe123qwedfs9878678sd }

PUT - /api/v1/landlord/logoff	Logout from account	**Body: { token**: SessionToken}	Confirmation Message	200	500	PUT https://52.139.9.83:80/api/v1/landlord/logoff	{"message": "Successfully logged out."}
POST - /api/v1/landlord/property	CREATE landlord property	**Body: { street_address:** string, **city:** string, **pincode:** string,**availability:** string, **beds:** string, **baths:**string , **rent:** string, **facilities:** list **}**	{ “property_id”: string }	200	500	POST https://52.139.9.83:80/api/v1/landlord/property	{"property_i": "P_dca578c5-6e24-48dc-8359-50cec9d239dd"}
DELETE - /api/v1/landlord/delprop/<prop_id>	DELETE landlord property		Confirmation message	200	500	DELETE https://52.139.9.83:80/api/v1/landlord/delprop/P_224c08c8-3781-4d5e-b0b4-851e11937a10	{ Message:”Property has been deleted” }

PUT- /api/v1/landlord/resolve_req/<query_id>	Landlord resolves service request	**{ "property_id"** : "P_224c08c8-3781-4d5e-b0b4-851e11937a10", **"tenant_id" :** "T_tavleen", **"resolution"**: "Replaced the lights", **"resolved"**: true **}**	Confirmation message	200	500	PUT https://52.139.9.83:80/api/v1/landlord/resolve_req/Q_d3ee95a1-a28f-419c-96fa-02a6c805bfc7	{ “query_id”:”Q_d3ee95a1-a28f-419c-96fa-02a6c805bfc7” }

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,**contact**: int **}**	{ “user_id”: “T_”+username }	200	500	POST https://52.139.9.83:80/api/v1/tenant/	{ user_id: “T_jessica”}

PUT - /api/v1/tenant/<user_id>	UPDATE tenant account	**Body: { email:** string, **fname:** string, **lname:** string, **password:**string,**contact**: int **}**	OK response	200	500	POST https://52.139.9.83:80/api/v1/tenant/<user_id>	{ Message: "Account has been updated"}
DELETE - /api/v1/tenant/<user_id>	DELETE tenant account		JSON of response from aws	200	500	DELETE https://52.139.9.83:80/api/v1/tenant/	{ Message: User Deleted }
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://52.139.9.83:80/api/v1/tenant/login	{"jwt": 123lkj312hlkaluw09789kn }

PUT - /api/v1/tenant/logoff	Logout from account		Confirmation Message	200	500	PUT https://52.139.9.83:80/api/v1/tenant/logoff	{"message": "Successfully logged out."}
POST - /api/v1/tenant/service_req	Tenant creates a new service request for his property	{"**property_id**": property_id, "**query**": query_id }	query_id for the service request would be returned	200	500	POST https://52.139.9.83:80/api/v1/tenant/service_req	{"query_id": "Q_6691f77d-bc21-4057-93b3-a348f29295e3"}
PUT - /api/v1/tenant/service_req_update/<query_id>	Tenant can update his/her service request	{"**property_id**": property_id, "**query**": query_id }	query_id for the service request would be returned	200	500	PUT https://52.139.9.83:80/api/v1/tenant/service_req_update/Q_d3ee95a1-a28f-419c-96fa-02a6c805bfc7	{"query_id": "Q_6691f77d-bc21-4057-93b3-a348f29295e3"}

PUT /api/v1/tenant/apply/<user_id>	Tenant applies for the property	{"**prop_id**": property_id}	Confirmation Message	200	500	https://52.139.9.83:80/api/v1/tenant/apply/T_tavleen	{"message": "Property Assigned"}

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/list_prop/<city_id>	List all property ids for a city		JSON of Property ids
			
			{Properties:<list_of_property_ids> }
				200	500	GET https://52.139.9.83:80/api/v1/property/city/Burnaby	{"properties": ["P_ccc7e307-4f1b-4eb0-872d-7a23eb024f3f", "P_1e137c20-1850-4faa-90ec-687d86d7d053", "P_67bc20f8-ac4c-4a6a-aacc-dc9903dd03d4"}
							
							}

GET - /api/v1/property/user_prop<user_id>	View all property ids for a user		JSON of Property ids {Properties:<list_of_property_ids> }	200	500	GET https://52.139.9.83:80/api/v1/property/user>prop/L_mike>	{ "property_details": ["P_1e137c20-1850-4faa-90ec-687d86d7d053","P_ac176033-a259-455b-a382-1af50047dc0b", "P_035c9965-4e69-4be5-b238-8164b8c4bcd4" } }

GET - /api/v1/property/<prop_id>	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://52.139.9.83:80/api/v1/property/P_02dbab20-14da-45dd-b0a8-16a93d6a86f8	{"property_details": { "availability": false, "baths": "3", "beds": "5", "city": "Surrey", "facilities": "[Garden,Backyard, Parking]", "pincode": "XT2UKJ", "property_id": "P_02dbab20-14da-45dd-b0a8-16a93d6a86f8", "rent": "1968", "street address": "191 Manor Street","user_id": "L_mike"}}

GET - /api/v1/property/service_req/	View all service requests created for landlord's properties	**Body:{ user_id**: "L_"+username}	JSON of Service request details of a property	200	500	GET https://52.139.9.83:80/api/v1/property/service_req/	{ "service_requests": [{"property_id": "P_2c5bba3d-99bf-4ab9-8340-f96efa014742", "service_request": { "T_tavleen": {"Q_261cf1f5-8aa4-46f1-9be0-e98ef87d85ba": { "query_str": "Bulb is not working",

							"resolved": false }
							}}}]}

Database Schema (DynamoDB)

Table: user_details-teamA756

Tag	**Value**	**Comment**
user_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	Tenant/Landlord contact number
properties	list	Properties owned by landlord stored as a list of property_ids

Table: property-teamA756

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

Table: service_requests-teamA756

Tag	Value	Comment
property_id	string	Property ID **(KEY)**
user_id	string	Tenant ID who raised the request (<t_username>)
query_id	string	Request ID
query_str	string	Message for service request
resolved	bool	Flag to check if the request is resolved or not

Table: city-teamA756

Tag	**Value**	**Comment**
city_id	string	City ID **(KEY)**
properties	string	Property ID is appended into each City’s value

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/Project 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.

Observations by team members in alignment with the readings:

- We learnt the importance of disaster recovery failure testing. It’s very important to find vulnerabilities in critical systems by intentionally causing failure in them. We experienced the same while doing failure testing for our services. We were able to analyse the system’s fault tolerance by applying Circuit breakers and Fault injection techniques in our services.
- We understood the benefit of following Agile principles. We divided the project work in smaller chunks and distributed among team members. This helped us in better management and allowed the team members to implement changes on short notice. It also helped us identify and predict risks and plan to ensure the smooth running of the project.
- By following the artifacts of the Scrum model, we were able to operate better in a coordinated manner through the use of Product Backlog, Sprint Backlog and Milestones. This helped us work based on deliverable deadlines to implement new features and fix bugs in the application.
- We understood the importance of using distributed systems such as Kubernetes for running the services. We were able to see the high availability and scalability in our services using Kubernetes. We spawned and configured new pods in our service and it helped us in the process of scaling, managing, updating and removing containers.
- Using version control, we were able to work separately on different parts of the project and manage the history of the code. All the team members created branches based on the issues they were working on which helped us work independently and write code without interfering with one another's work.

Section 4 - Analysis

Microservices architecture:

The independent deployability of all the services allowed for more autonomy and supported CI/CD which in turn encouraged adoption of Agile practices. Microservices are easily scalable which allows for easier load management and efficient resource consumption. For example, easy deployment of another replica (to implement round-robin) allows for easy traffic routing and management of requests in case of a faulty service. It was also relatively easier to isolate faults in microservices compared to monolithic applications.

Orchestration (k8s) and service mesh (istio)

Using Kubernetes allowed us to leverage immensely greater computation power than allowed by our local systems. It also significantly increased our productivity. An application deployed on a k8s orchestration could be accessed by other teammates simply by sharing the external IP for the cluster. Containers were automatically maintained (restarted/killed/replaced as needed), making it seamless to use. There was abundant documentation for Kubernetes with outstanding community and application support. The Azure portal gave us quick and easy insights into the orchestration and allowed us to interact with the orchestration with ease.

Istio was extremely useful in Failure Analysis and remediation. The documentation provided detailed information and step-by-step guide into Traffic Management with rich routing rules, retries, failovers, fault injection, etc. It was also exceptionally useful in enabling observability and providing metrics, logs and traces for all traffic enabling us to understand and visualize the internal working of our application.

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 the following test.

- **Basic Positive Tests -** Executing API calls with **valid required parameters**.

To scope this project, we left out the fuzzy testing for the following types -

- **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.

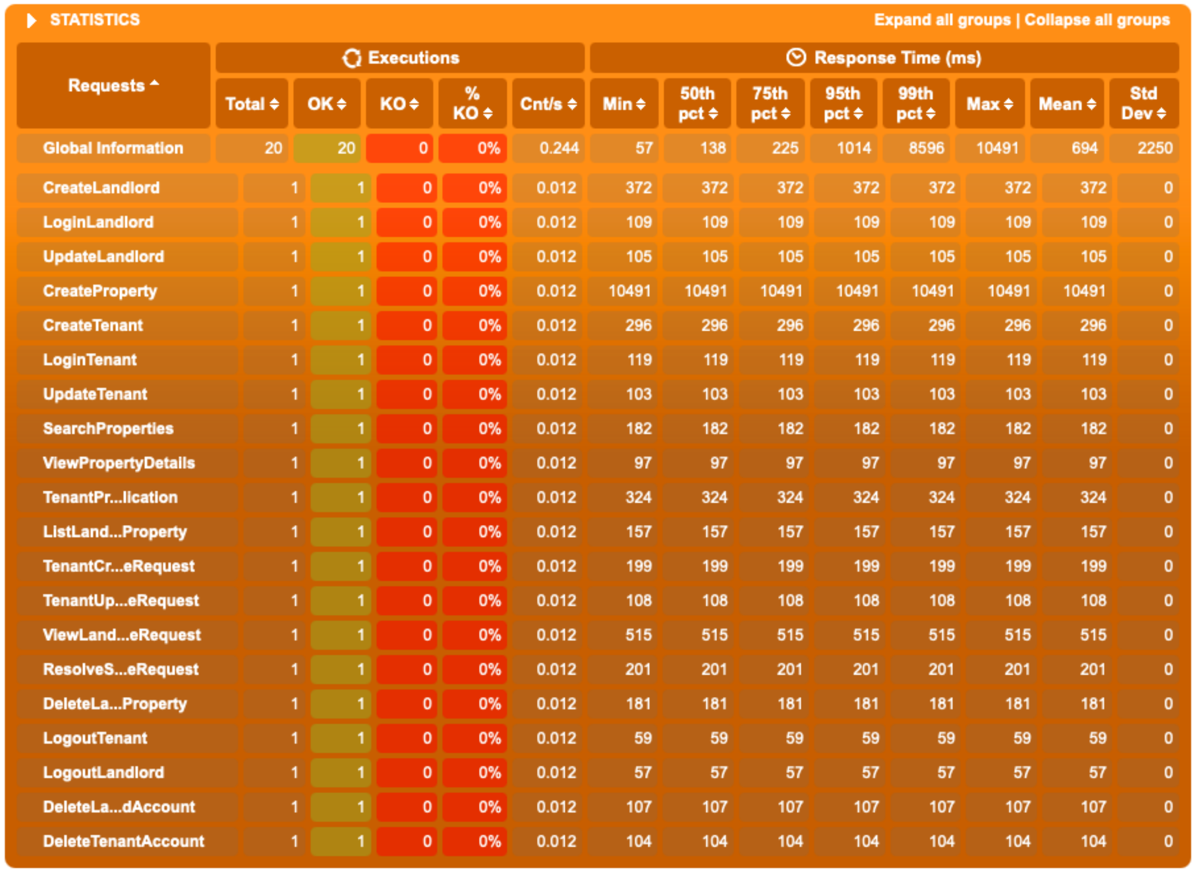
The coverage simulation exercise allowed us to validate the functional working of our API. We tested a variety of scenarios to ensure that the flow of the application does not break. The coverage simulation ran in parallel with creation of APIs. Each team member co-wrote the simulation scripts as soon 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.

API Call	API Description	Test Action Description
POST /api/v1/landlord/	Create Landlord Account	Landlord can create a new account
PUT /api/v1/landlord/login	Landlord Login account	Landlord can login into the account
PUT /api/v1/landlord/${l_user_id}	Update Landlord Account	Landlord can update his account
POST /api/v1/landlord/property	Create Property by Landlord	Landlord can create a property. This action results in updating the property table with property details, user details table by attaching a property id to the landlord’s account and city table by appending property ID to the city table
POST /api/v1/tenant/	Create Tenant Account	Tenant can create a new account
PUT /api/v1/tenant/login	Tenant Login account	Tenant can login into the account
PUT /api/v1/tenant/${t_user_id}	Update Tenant Account	Tenant can update his account
GET /api/v1/property/list_prop/${city_id}	List Properties by City	Tenant can list the properties in a city
GET /api/v1/property/${prop_id_response}	View Property Details	User can view details of a property
PUT /api/v1/tenant/apply/${t_user_id}	Tenant Property Application	Tenant applies for a property
GET /api/v1/property/user_prop/${l_user_id }	List Landlord Properties	Landlord can list all of his properties

POST /api/v1/tenant/service_req	Tenant Create Service Request	Tenant can create a service request for his rented accomodation
PUT /api/v1/tenant/service_req_update/${query_id_response}	Tenant Update Service Request	Tenant can update the service request he created previously
GET /api/v1/property/service_req	Landlord View Service Request	Landlord can view all the service requests for his/her property
PUT /api/v1/landlord/resolve_req/${query_id_response}	Landlord Resolves Service Request	Landlord can resolve a particular service request
DELETE /api/v1/landlord/delprop/${prop_id_response}	Delete Property	Landlord can delete the listed property
PUT /api/v1/tenant/logoff	Tenant Logoff	Tenant can logout of the account
PUT /api/v1/landlord/logoff	Landlord Logoff	Landlord can logout of the account
DELETE /api/v1/landlord/${l_user_id}	Delete Landlord Account	Landlord deletes account
DELETE /api/v1/tenant/${t_user_id}	Delete Tenant Account	Tenant deletes account

Section 4.1.1 Coverage Results

Gatling simulation was successfully run by hitting all the APIs created under the Landlord, Tenant and Property service. The following are the results of our Positive Testing Simulation:


STATISTICS

Expand all groups | Collapse all groups

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	20	20	0	0%	0.244	57	138	225	1014	8596	10491	694	2250
CreateLandlord	1	1	0	0%	0.012	372	372	372	372	372	372	372	0
LoginLandlord	1	1	0	0%	0.012	109	109	109	109	109	109	109	0
UpdateLandlord	1	1	0	0%	0.012	105	105	105	105	105	105	105	0
CreateProperty	1	1	0	0%	0.012	10491	10491	10491	10491	10491	10491	10491	0
CreateTenant	1	1	0	0%	0.012	296	296	296	296	296	296	296	0
LoginTenant	1	1	0	0%	0.012	119	119	119	119	119	119	119	0
UpdateTenant	1	1	0	0%	0.012	103	103	103	103	103	103	103	0
SearchProperties	1	1	0	0%	0.012	182	182	182	182	182	182	182	0
ViewPropertyDetails	1	1	0	0%	0.012	97	97	97	97	97	97	97	0
TenantPr...lication	1	1	0	0%	0.012	324	324	324	324	324	324	324	0
ListLand...Property	1	1	0	0%	0.012	157	157	157	157	157	157	157	0
TenantCr...eRequest	1	1	0	0%	0.012	199	199	199	199	199	199	199	0
TenantUp...eRequest	1	1	0	0%	0.012	108	108	108	108	108	108	108	0
ViewLand...eRequest	1	1	0	0%	0.012	515	515	515	515	515	515	515	0
ResolveS...eRequest	1	1	0	0%	0.012	201	201	201	201	201	201	201	0
DeleteLa...Property	1	1	0	0%	0.012	181	181	181	181	181	181	181	0
LogoutTenant	1	1	0	0%	0.012	59	59	59	59	59	59	59	0
LogoutLandlord	1	1	0	0%	0.012	57	57	57	57	57	57	57	0
DeleteLa...dAccount	1	1	0	0%	0.012	107	107	107	107	107	107	107	0
DeleteTenantAccount	1	1	0	0%	0.012	104	104	104	104	104	104	104	0

Section 4.2 - Load Testing Simulation

How did the load simulation help with testing of your completed system?

- We were able to determine the maximum operating capacity of our services.
- It helped us to determine whether the current infrastructure is sufficient to run the application
- Number of concurrent users that an application can support, and scalability to allow more users to access it.
- The load testing allowed us to identify bottlenecks in our system. As we have the majority of our requests being directed to property service, with property service triggering a set of cascading requests to DB, we were able to identify the property service as a potential bottleneck.

What types of failures did you simulate and what were the outcomes?

- We increased the number of users and tested the capacity of the services at peak performance. We observed that the database service started giving errors and analyzed the system's behavior. We configured the read and write capacity values of the DynamoDB database configuration which increased the efficacy of our system to process more requests successfully.

How did your application respond to various disturbances to the network?

The application response can be seen in the kiali graph before. Our API's have a great level of complexity where calls get routed and result in simultaneous updates to multiple tables. This complex network routing can be observed in the kiali graph below.

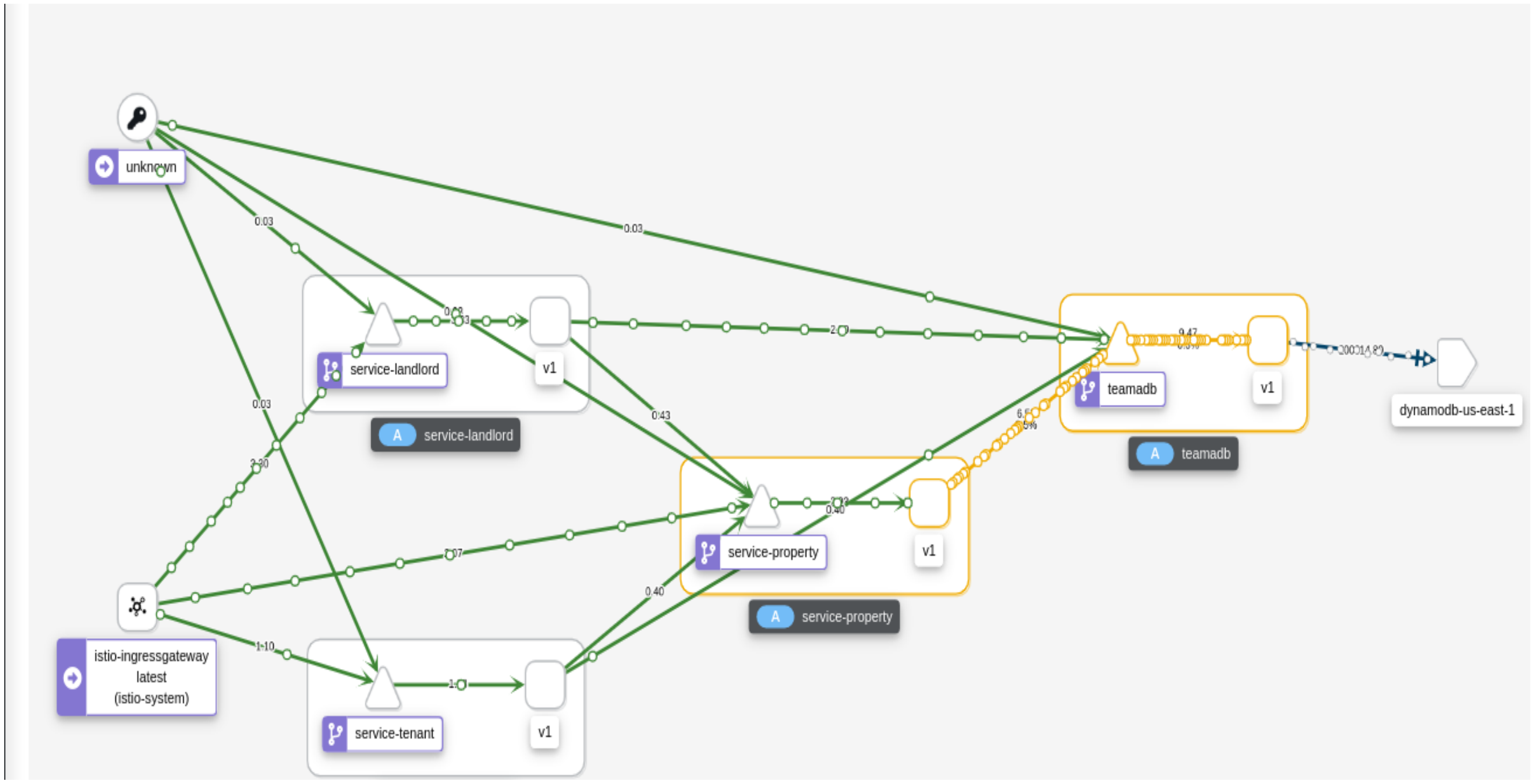
We performed operations that covered our API from end-to-end. The operations were scenarios on which we ran our load simulation. The operations were:

Landlord Create Property:
1. Landlord Login: Landlord logs in to his account to create a new property.
2. Create Property: The creation of property by a landlord results in simultaneous updates to various tables in our application. The property table gets updated with the newly created property details, the property gets attached to the respective landlord. The property id also gets appended to the city in the city table to perform a search property in the city.
3. Landlord Logoff: Post completion of operation, landlord logs off from the application.

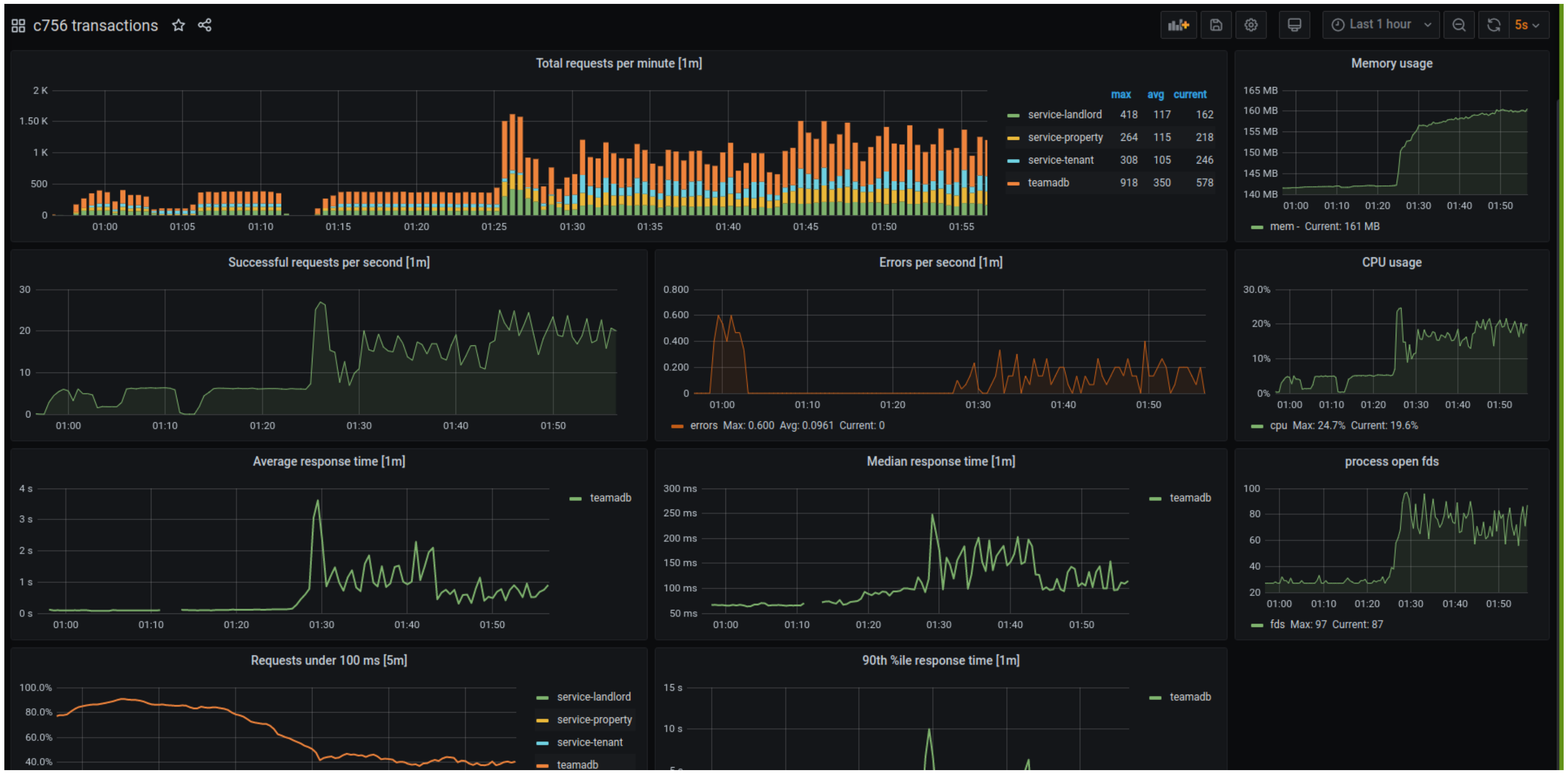
Landlord View property details:
1.Landlord Login: Landlord logs in to view his available properties.
2.Landlord views property: Landlord views his properties.

Tenant Creates Service Request:
1.Tenant Login: Tenant logs in to create a service request
2.Tenant Create Service Request: Tenants can create a service request for their property. We enable an upsert method to ensure that a tenant can create multiple requests for their rented accommodation as well as other tenants within the same property can create a hierarchy of service requests.
3. Tenant Logoff: The delete property request

This complex network of calls surely creates a busy network which results in some of our requests failing as we ramp up the user load. This can be observed in the Kiali graph below which shows some of the requests failing for the service property entity.


unknown
0.03
0.03
0.03
2.30
service-landlord
v1
service-landlord
0.43
2.07
teamadb
v1
teamadb
9.47
6.5%
dynamodb-us-east-1
service-property
v1
service-property
0.40
0.40
2.07
istio-ingressgateway
latest
(istio-system)
1.10
service-tenant
v1

Looking at the Grafana dashboard, we can observe the following:


c756 transactions
Last 1 hour
5s
Total requests per minute [1m]
Memory usage
mem - Current: 161 MB
Successful requests per second [1m]
Errors per second [1m]
errors Max: 0.600 Avg: 0.0961 Current: 0
CPU usage
cpu Max: 24.7% Current: 19.6%
Average response time [1m]
teamadb
Median response time [1m]
teamadb
process open fds
fds Max: 97 Current: 87
Requests under 100 ms [5m]
service-landlord
service-property
service-tenant
teamadb
90th %ile response time [1m]
teamadb

	max	avg	current
service-landlord	418	117	162
service-property	264	115	218
service-tenant	308	105	246
teamadb	918	350	578

<u>Requests under 100ms:</u> As the system was subjected to a single user load concurrently, the system had more than 90% requests processed and persisted in under 100ms. There is a drop as the number of concurrent users are increased from 2, 5 and 10 respectively. The system has errors once the number of concurrent users performing DML operations on the DB increases beyond 10.

Successful Requests per second: The average number of successful requests per second is around 20. Increasing the DB capacity to 20 helps in general stability of our application.

Errors per second: The sum of the rate of errors(where status code is not 200) increases once the number of concurrent users is increased beyond 10.

90% Percentile response times: The response time for more than 90% requests is around 2s. Thus, the application is stable and more than 80% of the requests are responded to in time.

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.

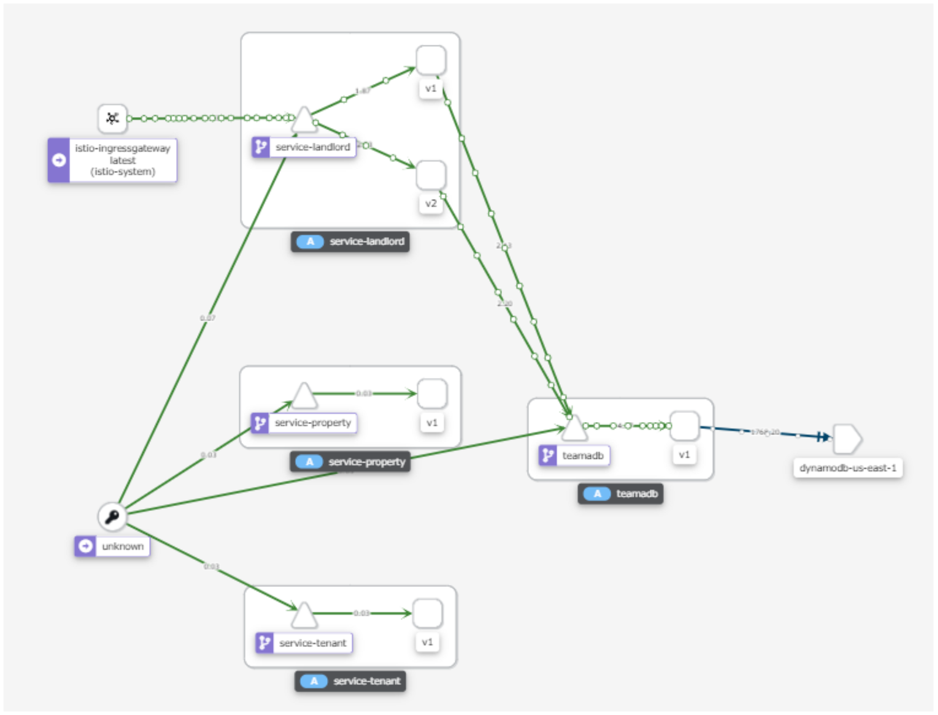
Section 4.3 - Failure Analysis and Remediation

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.

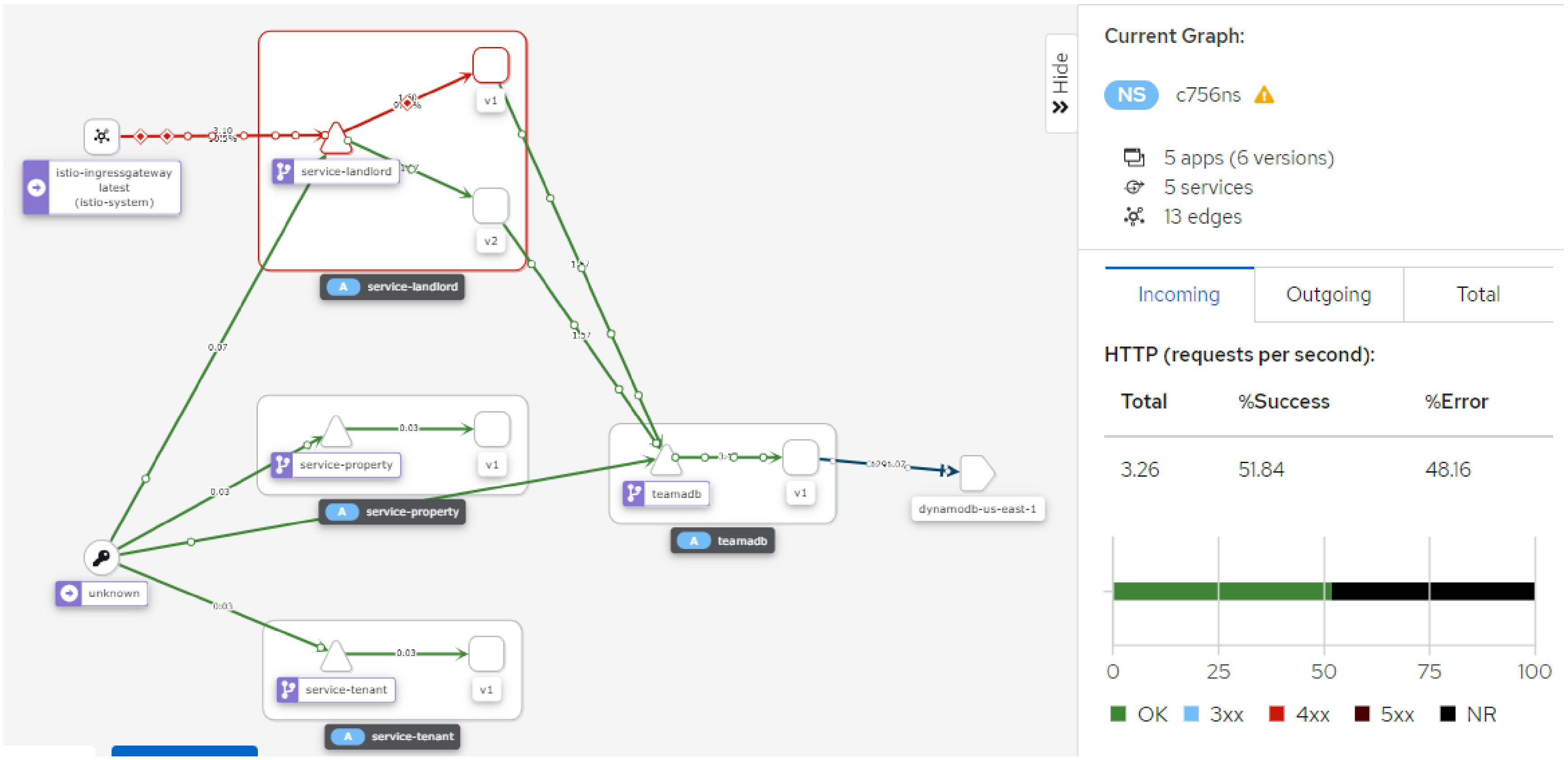
Experiment	Hypothesis	Observation	Method of remediation	Conclusion
Blue/Green Deployment	Introduce failures in one version(V1) of the service and make all the requests fail.	All the requests going from the landlord service from version V1 failed.	We introduce a new version and shift the traffic from V1 to V2 and then delete the version V1	We were able to successfully deploy a new version and shift the entire traffic without causing any downtime in the system.
Fault Injection: HTTP Error 503	Introduce failures in the Landlord Service by modifying code to return httpStatus 503 for 50% of the request	Approximately 50% of the requests in the Landlord Service V1 failed with HTTP Error Code 503.	Introduce a fix to set retry attempts to 3.	A failed request is retried 3 times. Since 50% of the load is round-robined to Landlord service V2 which has no error, all requests are successfully completed with 0% error.
Fault Injection: Delay	Introduce a 10 sec delay in the response time of the Landlord Service (wait time)	This significantly slowed down the time taken to get a response from the Landlord service V1.	Introduce a fix to set timeout to 1 second after which a request fails.	All requests on delayed Landlord service V1 failed and all requests on error-free Landlord Service V2 succeeded.

Circuit Breaker	Introduce a delay in the application and add multiple users accessing the service concurrently	This queued requests and made the Landlord Service V1 significantly slower.	Introduce a circuit breaker (fail fast) that only allows 1 pending request at a time and fails the rest.	A majority of the request on the slow Landlord Service V1 failed due to queueing up.

System State (Delay in S1)



System State (With Timeout)



Current Graph:

NS c756ns ⚠

5 apps (6 versions)
5 services
13 edges

Incoming | Outgoing | Total

HTTP (requests per second):

Total	%Success	%Error
3.26	51.84	48.16

🟩 OK 🟦 3xx 🟥 4xx ■ 5xx ■ NR