CMPT 756.203 T4-Wednesday Term Project Submission

Team	T4-Wednesday		
Section	G100		
Team Member Name	Kunal Niranjan Desai	Rohan Harode	Juan Ospina
Email (SFU)	kdesai@sfu.ca	rharode@sfu.ca	jospina@sfu.ca
Email (G-Suite/Gmail)	kdesai1505@gmail.com	harode48@gmail.com	juandavidospina@gmail.com
Github ID	kunaldesai97	rohanharode	juandavidospina
Additional Notes			

Section 1 - Problem Statement

Problem Domain & Application

Online Bill Payment System: "My Bill Payments"

Our client, a recognized Fintech named "ABCFintech," wants to offer its current and potential customers an application to pay their bills online. This application is called "My Bill Payments". Through this application, a customer gets a centralized platform to pay all their bills from different companies like BC Hydro (Electricity), Rogers (cellphone), Telus (Internet) (from now on, named "billers") as well as manage them, thus saving time and avoiding overdue balances and late fees.

For example, a person named John Doe wants to pay three bills at the end of the month: BC Hydro, Cellphone (Rogers), and Internet (Telus). John will be able to log in on our application, and through the "Bill retrieval" functionality, he can look up the details of each bill, using as input the biller name and the bill identifier, e.g., Biller name: "Rogers" and Bill ID: 12345. Assuming that the bill exists, our system will return the bill's relevant information: Biller name, Bill identifier, amount to pay, and due date. Then, John can pay this bill using his credit card. Hence, John will enter its credit card information into the system: credit card number (16 digits), cardholder name, expiration year-month, and the three-digit security code. In case that John has funds available to cover this payment, our system will flag the bill as paid (preventing that it can be paid again by another user) and register the payment information (credit card first four and last four digits, cardholder name and expiration year-month). The user can repeat this process as many times as the number of bills. Also, if John wants to review his paid bills, our system will provide a functionality to review all his past payments (payments history).

Specification of REST API (Microservices Contract)

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

API	**Description**
/api/v1/user	Users service URL
/api/v1/bill	Bills service URL
/api/v1/billers	Billers service URL

Service: Users

Version: v1
Visibility: Private
Domain: Users
Serialized Data/Content-Type: json

API	Description	Request Body/Parameters	Response Body	HTTP Response Code	Error Codes	Request Example	Response Example
PUT - /api/v1/user/login	Login	JSON:{"uname" : user_name, “pword”: password}	Hash of user context suitable for passing to other calls	200	500	PUT https://host:5000/api/v1/user/login	{“UserContext”: “”}
PUT - /api/v1/user/logoff	Logoff	JSON: {"jwt": token}	None	200	500	PUT https://host:5000/api/v1/user/logoff	{ Message: ok }
PUT - /api/v1/user/	UPDATE one user	Body: { uname: string, pword: string, Fname: string, Lname: string, Email: string, Secquestion: string,	OK response	200	500	PUT https://host:5000/api/v1/user/	{ Message: ok }

		Secanswer:string }					
POST - /api/v1/user/	CREATE one user	Body: { uname: string, Fname: string, Lname: string, Email: string, Secquestion:string, Secanswer:string } Params: None	OK response	200	500	POST https://host:5000/api/v1/user	{ ResponseMetadata: { … etc. } }
DELETE - /api/v1/user/	DELETE one user by username		JSON of response from aws	200	500	DELETE https://host:5000/api/v1/user/	{ ResponseMetadata: { … etc. } }
GET - /api/v1/user/<uname>	GET one user by username		JSON of User entity	200	500	GET https://host:5000/api/v1/user?user=johndoe	{User Object}

GET - /api/v1/user?userkey	GET user by key	Query Params: userkey:string	JSON of User entity	200	500	GET https://host:5000/api/v1/user?userkey=email:johndoe@gmail.com	{User Object}

Service: Bill

Version: v1
Visibility: Private
Domain: Bills
Serialized Data/Content-Type: json

API	Description	Request Body/Parameters	Response Body	HTTP Response Code	Error Codes	Request Example	Response Example
GET - /api/v1/bill/	Retrieve one bill	Param: bill_id	None	200	500	GET https://host:5001/api/v1/bill/	{ bill_id: string, biller_id:string, amount:float, due_date: date, bill_paid:boolean}
POST - /api/v1/bill/	Insert one bill	Body: { user_id: string, biller_id:string, bill_amount:float, due_date:date, bill_paid: boolean	OK response	200	500	POST https://host:5001/api/v1/bill/	{ Message: ok }

		(default: False)} User-context token					
DELETE - /api/v1/bill/	DELETE one bill	Param: bill_id	JSON of response from aws	200	500	DELETE https://host:5001/api/v1/bill/	{ ResponseMetadata: { … etc. } }
PUT - /api/v1/bill/pay/<bill_id>	Pay bill	Body: {“cc_number”:, cc_exp_dat: String, payment_date: String} Header: User-context token	OK response	200	500	PUT https://host:5000/api/v1/bill/pay	{ Message: ok }

GET - /api/v1/bill/<uname>	GET bills by username	Query Params: username:string	JSON of Bill entity	200	500	GET https://host:5000/api/v1/bill/johndoe	{Bill Object}

Service: Biller

Version: v1
Visibility: Private
Domain: Billers
Serialized Data/Content-Type: json

API	Description	Request Body/Parameters	Response Body	HTTP Response Code	Error Codes	Request Example	Response Example
GET - /api/v1/biller/	Retrieve one biller	Param: biller_id	None	200	500	GET https://host:5001/api/v1/biller/	{ biller_id: string, biller_name:string, biller_description:string}
POST - /api/v1/biller	Insert one biller	Body: { biller_id: string, biller_name: string, biller_description:string }	OK response	200	500	POST https://host:5001/api/v1/biller/	{ Message: ok }

DELETE - /api/v1/biller/	DELETE one biller	Param: biller_id	JSON of response from aws	200	500	DELETE https://host:5001/api/v1/biller/	{ ResponseMetadata: { … etc. } }

Service: db

Version: v1
Visibility: Private
Domain: Datastore
Serialized Data/Content-Type: json

API	Description	Request Body/Parameters	Response Body	HTTP Response Code	Error Codes	Request Example	Response Example
GET - /api/v1/datastore/read	Read an object	Param: object-type, object-key	JSON of aws response	200	500	GET https://host:5002/api/v1/datastore/read?objtype=user&objkey=johndoe	{ "Count": 1, "Items": [{ "uname": "johndoe", "fname": "John", "lname": "Doe", "email": "johndoe@gmail.com", “secquestion”:”Name of your first pet?”, “secanswer”:”Lucky” }], "ResponseMetadata": { "HTTPHeaders": { "connection": "keep-alive", "content-length": "165", "content-type": "application/x-amz-json-1.0", "date": "Sat, 12 Sep 2020 18:16:15 GMT", "server": "Server", "x-amz-crc32": "196980578", "x-amzn-requestid": "AOGKN903DF66VLU3GEBEEO8DK3VV4K

							QNSO5AEMVJF66Q9ASUAAJG" }, "HTTPStatusCode": 200, "RequestId": "AOGKN903DF66VLU3GEBEEO8DK3VV4KQNSO5AEMVJF66Q9ASUAAJG", "RetryAttempts": 0 }, "ScannedCount": 1 }
POST - /api/v1/datastore/write	Write an object	Body: objtype, object-key(s)	ID of new entity	200	500	POST https://host:5002/api/v1/datastore/write	{ "uname": "johndoe" }
DELETE = /api/v1/datastore/delete	Delete an object	Param: objtype, object-key	JSON of aws response	200	500	DELETE https://host:5000/api/v1/datastore/delete?objtype=user&objkey=johndoe	{ "ResponseMetadata": { "HTTPHeaders": { "connection": "keep-alive", "content-length": "2", "content-type": "application/x-amz-json-1.0", "date": "Sat, 12 Sep 2020 18:13:04 GMT", "server": "Server", "x-amz-crc32": "2745614147", "x-amzn-requestid": "N7R6LO93FFBDH1A5GRRL55LBS7VV4KQNSO5AEMVJF66Q9ASUAAJG" }, "HTTPStatusCode": 200, "RequestId": "N7R6LO93FFBDH1A5GRRL55LBS7VV4KQNSO5AEMVJF66Q9ASUAAJG", "RetryAttempts": 0 } }

PUT - /api/v1/datastore/update	Update an object	Params: objtype, objkey Body: object-key(s) Example: email	JSON of aws response	200	500	PUT https://host:5002/api/v1/datastore/update?objtype=user&objkey=email:johndoenew@gmail.com	{ "ResponseMetadata": { "HTTPHeaders": { "connection": "keep-alive", "content-length": "2", "content-type": "application/x-amz-json-1.0", "date": "Sat, 12 Sep 2020 18:13:04 GMT", "server": "Server", "x-amz-crc32": "2745614147", "x-amzn-requestid": "N7R6LO93FFBDH1A5GRRL55LBS7VV4KQNSO5AEMVJF66Q9ASUAAJG" }, "HTTPStatusCode": 200, "RequestId": "N7R6LO93FFBDH1A5GRRL55LBS7VV4KQNSO5AEMVJF66Q9ASUAAJG", "RetryAttempts": 0 } }

Database Schema (DynamoDB)

Table: Users

Tag	Value	Comment
lname	string	Last name of user
fname	string	First name of user
uname	string	Email ID of user
password	string	User’s account password
security_question	string	Security question in case user forgets password
security_answer	string	Answer to the security question
user_id	string	Unique id generated by DynamoDB

Table: Bills

Tag	Value	Comment
bill_amount	decimal	Amount billed
bill_paid	boolean	true/false
due_date	string	Due date for the bill
payment_date	string	Date of the bill payment

cc_first_four_digits	integer	Identification of credit card franchise: Visa, MasterCard etc
cc_last_four_digits	integer	Credit card information for troubleshooting
cc_exp_date	string	Expiry date of credit card
auth_code	integer	Authorization Code generated after payment confirmation
user_id	string	Username
biller_id	string	Unique id generated by DynamoDB
bill_id	string	Unique id generated by DynamoDB

Table: Billers

Tag	**Value**	**Comment**
biller_name	string	Name of the billing company
active	boolean	true/false
biller_description	string	Description of the billing company
biller_id	string	Unique id not interpreted by DynamoDB

Section 2 - Github Repo Guide

Path	**Note**
/docs	Information of various services
/docs/user_service	Details of users service and functionalities
/docs/bill_service	Details of bills service and functionalities
/docs/biller_service	Details of billers service and functionalities
/docs/minikube	Minikube steps, commands
/docs/docker	Docker steps, commands
/docs/AKS	Azure Kubernetes commands
/code	Code for API calls of different services
/code/users	Code for API calls of users services
/code/bills	Code for API calls of bills services
/code/billers	Code for API calls of billers services
/IaC	Infrastructure as a Code directory
/IaC/cloudformation	Cloudformation stacks
/IaC/cluster	Cluster creation
/IaC/k8s	Service gateway

Section 3 - Reflection on Development

We adopted the idea of creating a product backlog to have a clear understanding of all the services we plan to implement for the project. We also had weekly meetings for 30 minutes each to catch up on the progress that will help us stay on track with our project plans, collaborate with others and brainstorm regarding the application and REST APIs. Since we are a team of 3 (with no scrum master or product owner), each of us took responsibility to make sure we were focusing on the bigger picture of the project while undertaking tasks. We have already started the Scrum process by creating a Kanban dashboard on Github and aiming to create stories and tasks after the feedback of the 1st milestone. We are also maintaining a shared Google Doc to suggest and incorporate changes to the proposed application, architecture, REST API's and resolving them to promote a continuous cycle of improvement.

The readings which were most helpful to us were R2 (Scrum guide), R3 (Epics, stories, tasks, subtasks), and R6 (Revision control systems). The scrum guide gave us a good understanding of how scrum works and methodologies. It motivated us to try it for the term project starting with the product backlog and weekly meetings proving beneficial to keep us on track and discuss the project requirements. The R3 reading introduced us to organizing tasks within the team and implementing the ideas on Github's dashboards for tracking progress. R6 reading helped us use Git for our tasks with individual branches and then review each of our code to finally piece together each of our contributions to the master for submission.

Our team performs well, adheres to the scrum principles, makes good progress, and handles tasks regularly. In our professional experience, we belonged to a larger team with a scrum master to supervise the progress. The unavailability of the product owner or scrum master added a bit more load on each of us, but we managed to take responsibility for our tasks and bring it together for the project. We believe that scrum success depends a lot on the team members if they actively participate in the meetings, collaborate well, etc. Consequently, we will ensure that each of us is proactive in his work and makes valuable contributions at every stage of the development lifecycle.