System Requirements Specification Index

For

Django Rest-API
Credit Card Management System

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

TABLE OF CONTENTS

1. Project Abstract

Credit Card Management Application is Django RESTful application withSQLite database, where it allows customer to deposit or withdraw the money, and same would be processed by bank manager.

Following is the requirement specifications:

	Commercial Credit Card Management
Users	
1	Manager
2	Customers
Modules	
1	CustomerModel
2	CardTypesModel
3	RequestModel
4	LimitIncreaseRequestModel
5	CardDetailsModel
Manager Functionalities	
1	Add a card
2	List all card request
3	Accept a card request
4	Reject a card request
5	List all limit increase request
6	Accept an increase limit request
7	Reject an increase limit request
8	List all blocked cards
9	Unblock a card
Customer Functionalities	
1	List all types of cards

2	Register a customer
3	Apply for a card
4	Get status of card application
5	Block the card
6	Request to increase the limit

2 Assumptions, Dependencies, Risks / Constraints

- If customer requestId is not valid, application should throw a custom exception (IdDoesNotExist).
- While blocking or unblocking card, If cardId is not valid, application should throw a custom exception (CardIdDoesNotExist).
- While applying card if customer id and card id is invalid throw a custom exception (InvalidId).
- For increase limit request, if card is blocked throw a custom exception(CardBlocked)
- If a customer applies for same type of card more than once, throw a custom exception (AlreadyApplied).
- A customer can't apply for more than 3 cards, if such request is generated throw a custom exception (CanNotApply).

3 Business Validations

- Customer name max 50 characters.
- Email max 50 characters.
- Card type name max 50 characters.
- Card type description max 100 characters.
- Remark max 100 characters.
- Boolean filed default value should be False.

4 Rest Endpoints

Rest End-points to be exposed in the controller along with method details for the same to be created

Class Name	Method Name	Purpose Of Method
AddCardTypesView	post(self,request,format=None)	To add card types.

ListAllCardRequestsView	get(self,request,pk=None,format=None)	To list all requested cards.
ListLimitIncreaseRequestsView	get(self,request,pk=None,format=None)	To list all limit increase requests.
ListAllBlockedCardsView	get(self,request,pk=None,format=None)	To list all blocked cards.
UnBlockCardView	patch(self,request,pk,format=None)	To un block the card.
AcceptCardRequestView	post(self,request,pk,format=None)	To accept/save card request.
RejectCardRequestView	patch(self,request,pk,format=None)	To reject and update card request.
AcceptIncreaseCardLimitRequestView	patch(self,request,pk,format=None)	To accept limit increase request for the card.
RejectIncreaseCardLimitRequestView	patch(self,request,pk,format=None)	To reject limit increase request for the card.
CustomerRegistrationView	post(self, request,format=None)	To register the customer.
ListCardsAndStatusView	get(self,request,pk=None,format=None)	To list all types of cards and get status of card application.
BlockCardView	patch(self,request,pk,format=None)	To block the card.
RequestForIncreaseLimitView	post(self,request,format=None)	To request for increase limit of a card.
ApplyForCardView	post(self, request,format=None)	To apply for the card.
	put(self,request,pk,format=None)	To update the request for card.

Resources (Models)

Class	Description	Status
CustomerModel	o A model class for Customer. o It will map to the CustomerModel table.	Already implemented.
CardTypesModel	o A model class for CardTypes. o It will map to the CardTypesModeltable.	Already implemented.
RequestModel	o A model class for Request. o It will map to the RequestModel table.	Already implemented.
LimitIncreaseRequestModel	o A model class for Limit Increase Request. o It will map to the LimitIncreaseRequestModelta ble.	Already implemented.
CardDetailsModel	o A model class for CardDetailsRequest. It will map to the CardDetailsModeltable.	Already implemented.

Resources (Serializers)

Class	Description	Status
CustomerSerializer	A serializer for CustomerModel	Already implemented
CardTypesSerializer	A serializer for CardTypesModel	Already implemented
RequestSerializer	A serializer for RequestModel	Already implemented
LimitIncreaseRequestSerializer	A serializer for LimitIncreaseRequestModel	Already implemented
CardDetailsSerializer	A serializer for CardDetailsModel	Already implemented

Resources (Views)

Class	Description	Status
AddCardTypesView	A class for the post functionalitiy on **CardTypesModel**model**.**	To be implemented
ListAllCardRequestsView	A class for the get functionality on **RequestModel**model**.**	To be implemented
ListLimitIncreaseRequestsView	A class for the get functionality on **LimitIncreaseRequestModel**model**.**	To be implemented
ListAllBlockedCardsView	A class for the get functionality on **CardDetailsModel**model**.**	To be implemented
UnBlockCardView	A class for the patch functionality on **CardDetailsModel**model**.**	To be implemented
AcceptCardRequestView	A class for the post functionality on **RequestModel**model**.**	To be implemented
RejectCardRequestView	A class for the patchfunctionality on **RequestModel**model**.**	To be implemented
AcceptIncreaseCardLimitRequestView	A class for the patch f functionality on **LimitIncreaseRequestModel and CardDetailsModel**models**.**	To be implemented
RejectIncreaseCardLimitRequestView	A class for the patch functionality on **LimitIncreaseRequestModel**model**.**	To be implemented
CustomerRegistrationView	A class for the post functionality on **CustomerModel**model**.**	To be implemented

ApplyForCardView	A class for the put, post functionalities on **RequestModeland CardDetailsModel**models respectively**.**	To be implemented
ListCardsAndStatusView	A class for the get functionality on **CardDetailsModel**model**.**	To be implemented
BlockCardView	A class for the patch functionality on **CardDetailsModel**model**.**	To be implemented
RequestForIncreaseLimitView	A class for the post functionality on **CardDetailsModel**model**.**	To be implemented

Resources (Exceptions)

Class	Description	Status
InvalidId	Custom exception is thrown in case specified if customer id and card id is invalid.	Already implemented.
IdDoesNotExist	Custom exception is thrown in case specified customer request idis invalid.	Already implemented.
CardBlocked	Custom exception is thrown in case card is blocked.	Already implemented.
AlreadyApplied	Custom exception is thrown in casea customer applies for same type of card more than once.	Already implemented.
CanNotApply	Custom exception is thrown in case customer apply for more than 3 cards.	Already implemented.
CardIdDoesNotExist	Custom exception is thrown in case specified card Id is not valid.	Already implemented.

6 Execution Steps to Follow

1. All actions like build, compile, running application, running test cases will be through Command Terminal.
2. To open the command terminal the test takers, need to go to
   Application menu (Three horizontal lines at left top) -> Terminal ->New Terminal.
3. The editor Auto Saves the code.
4. If you want to exit(logout) and to continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.
5. These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.
6. To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN.
7. To test any UI based application the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.
8. Install 'djangorestframework' module before running the code. For this use the following command.

   pip install djangorestframework
9. Use the following command to run the server

   python3 manage.py runserver
10. Mandatory: Before final submission run the following commands to execute testcases

    python3 manage.py test cardapp.test.test_functional

    python3 manage.py test cardapp.test.test_exceptional

    python3 manage.py test cardapp.test.test_boundary

11. To test rest end points

    Click on 'Thunder Client' or use Ctrl+Shift+R ->Click on 'New Request' (at left side of IDE)

12. Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on "Submit Assessment" after you are done with code.

13. You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.

------- * --------