

**Blue Penguin – E-bidding Application
For Bidding System**

Version 3.0

Blue Penguin – E-bidding Application	Version: 3.0
Software Requirement Specification	Date: 12/12/24
Third Draft	

Revision History

Date	Version	Description	Author
12/11/24	3.0	Introduction and 1-1 Correspondence	Addina Rahaman
12/12/24	3.1	Contributions, Links, and Remarks	Addina Rahaman, Lily Minchala, Samiul Saimon

Blue Penguin – E-bidding Application	Version: 3.0
Software Requirement Specification	Date: 12/12/24
Third Draft	

Table of Contents

1. Introduction	4
2. Correspondence of System and System Specs	6
3. Contributions	10
4. Links and Other Credentials	11
5. Remarks	12
a. Database	12
b. Background Tasks/Logic	15
i. Suspensions	15
ii. VIP System	16

Blue Penguin – E-bidding Application	Version: 3.0
Software Requirement Specification	Date: 12/11/24
Third Draft	

1. Introduction

BluePenguin is a full-stack E-bidding platform built with a Django REST Framework backend, Supabase DBMS, and a React frontend. The system allows users to list items for auction, place bids, participate in transitions, and comment on items. It has a sophisticated user hierarchy defined by statuses such as Visitor, User, Superuser, and VIP, and a dual authentication system handled by both Django and Supabase. In addition to basic E-bidding features, we have also integrated a profile system where registered users can view and edit their profile, a feature that allows users to save an item, searching and filtering options, and an internal emailing system that sends email notifications on important bidding and transaction updates. Deadlines and other backgrounds are handled by celery-redis.

There are multiple packages and credentials required to run this website, all outlined in our [github repository](#). One must ensure they have Node, Python, and Postgresql installed in their local machine first. In the frontend directory, the following packages must be installed: npm, @supabase/supabase-js, axios, and react-router-dom. In the backend directory, the several packages outlined in requirements.txt must be installed. A separate .env file must be configured for the frontend and backend directories. The frontend .env file must contain (in quotes), the VITE_API_URL, VITE_SUPABASE_URL, and VITE_SUPABASE_ANON_KEY, all of which are obtained from the Supabase project credentials. In the backend .env file, the same credentials would be stored in SUPABASE_URL in SUPABASE_ANON_KEY respectively, as well as SUPABASE_SERVICE_ROLE_KEY. DB_NAME, DB_USER, DB_PASSWORD, DB_HOST, DB_PORT, can be obtained from the Supabase project. EMAIL_HOST_USER and EMAIL_HOST_PASSWORD are the official BluePenguin email credentials.

Our project stores profile pictures and item images in Google Cloud Platform, specifically Google Cloud Platform. The images have been converted to public URLs which are linked as image attributes for the Item table in our database. From the platform, one can create an Admin account with the correct roles and download the application credentials in JSON format. This file would be stored in the directory and the relative path would be assigned to GOOGLE_APPLICATION_CREDENTIALS in the backend .env file. In addition, GOOGLE_CLOUD_PROJECT_ID, and GOOGLE_CLOUD_STORAGE_BUCKET, are used to contain the Project ID and the name of the storage bucket that contains our images.

In order to run the project, one must run celery first. Open a new terminal in the backend directory and run: 'source venv/bin/activate' to activate the virtual environment. Open three

more separate terminals in the root directory; in the first one, run: 'redis-server'; in the second, run: 'celery -A backend worker -l info', and in the third, run 'celery -A backend beat -l info'. To run the system itself, in a backend terminal, run 'python3 manage.py runserver', and in a frontend terminal, run 'npm run dev'.

Blue Penguin – E-bidding Application	Version: 3.0
Software Requirement Specification	Date: 12/11/24
Third Draft	

2. Correspondence of System and System Specs

Project Specs	Corresponding Files	Completion Status
1. There are three types of users: visitors (V), users (U), and super-users (S).	/backend/api/models.py /backend/api/choices.py /frontend/src/pages/Profile/Profile.jsx	Finished
2. A V can browse the listing of items provided by U's and provide comments. V can apply to be U, which needs to be approved by S. To avoid automatic robots, a question should be answered by V in the application process to decide whether V is a human being: asking a random arithmetic question that robots cannot handle.	/backend/api/views.py /backend/api/serializers.py /backend/api/models.py /backend/api/admin.py /frontend/src/components/ItemCard/ItemCard.jsx /frontend/src/pages/ItemPage/ItemPage.jsx /frontend/src/components/Comment/Comment.jsx /frontend/src/pages/Auth/ApplyUser.jsx	Finished
3. A U can deposit/withdraw money from one's own account, can list any items/services s/he owns and the asking price for sale or rent, can list any items/services s/he needs and the price range s/he can accept; can bid on an item; he is interested	/backend/api/views.py /backend/api/utils.py /frontend/src/pages/PaymentsAndAddress/PaymentsAndAddress.jsx /frontend/src/pages/Items/AddItem.jsx /frontend/src/components/ItemCard/ItemCard.jsx /frontend/src/pages/ItemPage/ItemPage.jsx	Finished

in and deadline of the bid, the bidder U should have adequate money in the system to be qualified; the owner U decides to accept which U to sell/rent the item/service.		
4. Once the owner and buyer/renter U agree, the item is removed from the listing, and the corresponding price is transferred from the buyer/renter to the owner.	/backend/api/views.py /backend/api/utls.py /frontend/src/pages/Orders/NextActions.jsx	Finished
5. After the transaction is done, the buyer/renter can rate the owner: from 1-worst to 5-best. The owner can rate the renter after the transaction. U's do not know who rates them so they cannot retaliate or praise back. No one else can rate. A U can directly complain to S about the U's/he bought/rented items.	/backend/api/views.py /backend/api/models.py /frontend/src/pages/Profile/Profile.jsx /frontend/src/pages/Requests/Requests.jsx	Finished
6. A U whose rating less than 2 evaluated by at least three U's is suspended, which can only be active again by paying \$50 fine and is activated by S. A U whose average ratings (≥ 3 times) are less than 2, viewed as too mean, or greater than 4, viewed as too generous, is	/backend/api/tasks.py /backend/api/views.py /backend/api/models.py /backend/api/admin.py	Finished

suspended		
7.A U suspended three times is forced out of the system;A U can choose to quit from the system by applying to S.	/backend/api/models.py /backend/api/views.py /backend/api/admin.py /frontend/src/pages/Requests/Requests.jsx	Finished
8.A U is a VIP if 1) with more than \$5,000 in the account; 2) conducted more than 5 transactions; 3)without any complaints; A VIP receives a10% discount for every transaction; violations of the previous two conditions will move the VIP back to an ordinary U.VIP has all powers U enjoys.VIP who is supposed to be suspended will not be suspended but will be put back to an ordinary U.	/backend/api/models.py /backend/api/views.py	Finished
9.Provide a personalized GUI so that when a U login, information related to him/her based on her/his previous records is shown.	/backend/backend/middleware.py /backend/backend/settings.py /backend/api/models.py /backend/api/views.py /frontend/src/pages/Auth/Login.jsx /frontend/src/pages/Auth/Register.jsx /frontend/src/context/AuthContext.jsx /frontend/src/pages/Orders/Orders.jsx /frontend/src/pages/Orders/NextActions.jsx /frontend/src/pages/Orders/PendingBids.jsx /frontend/src/pages/Orders/SavedItems.jsx	Finished
10.One creative feature worthy of 10% of the system points is required, feature of special creativity will receive a special bonus (up to 10%).	Profile System: /backend/api/models.py /backend/api/views.py /backend/api/serializers.py /backend/api/utlis.py /frontend/src/pages/Profile/Profile.jsx /frontend/src/pages/Profile/EditProfile.jsx	Finished

	<p>Save an Item:</p> <p>/backend/api/models.py</p> <p>/backend/api/views.py</p> <p>/frontend/src/pages/Orders/SavedItems.jsx</p> <p>/frontend/src/pages/ItemPage/ItemPage.jsx</p> <p>Searching and Filtering:</p> <p>/backend/api/views.py</p> <p>/backend/api/filters.py</p> <p>/frontend/src/components/SearchBar.jsx</p> <p>/frontend/src/components/SearchResults.jsx</p> <p>/frontend/src/pages/Filter/Filter.jsx</p>	
--	---	--

Blue Penguin – E-bidding Application	Version: 3.1
Software Requirement Specification	Date: 12/12/24
Third Draft	

3. Contributions

Member	Contribution
Addina Rahaman	Created the entire backend, implemented background logic, connected our project to the database, and connected our project to our Google Cloud Storage bucket. Implemented credentials and permissions. Created Django models, and migrated models onto our database as tables. Created the backend API to be called. Implemented suspension system and VIP system. Created the Superuser's admin website. Implemented password hashing, email notifications, authorization middleware, file uploading to GCS, and celery-redis tasks. and other backend API endpoints (visible on /backends/api/views.py) that are called by frontend.
Samiul Saimon	Worked on frontend functionality by communicating properly with the backend through the endpoints and associated APIs utilizing said endpoints. Ensured Google Cloud storage received file uploads such as profile images, Django's APIs communicated successfully with React and Supabase and updated accordingly with certain actions such as posting a listing, updating address, and most intricately authentication with both Django and Supabase which had unique requirements to meet to ensure functionality. Additionally, worked on search bar functionality, navbar, page redirects, payments/addresses, apply to be user page, and refining various other details and minor styling.
Lily Minchala	Contributed to developing the project's front end by focusing on the design and functionality through CSS and JSX. Implemented key pages and components, including ItemPage, Orders, Requests, and Filters pages. Collaborated with team members to draft preliminary designs for "Blue Penguin" in Figma to serve as a guide for the website's graphical user interface (GUI). Additionally, organized sections of the project's dummy data that were placed in Supabase by using Google Sheets. Designed and illustrated the penguin graphics featured in the project to add some zest to the project's aesthetic.

Blue Penguin – E-bidding Application	Version: 3.1
Software Requirement Specification	Date: 12/12/24
Third Draft	

4. Links and Other Credentials

- A. **Github repository:** <https://github.com/iDaManall/BluePenguin>

- B. **Supabase Credentials** (to view our database):
 Email: officialbluepenguin@gmail.com
 Password: booyork70@Y
 Project Link: <https://supabase.com/dashboard/project/pzbntboptzxaiphgcb>

- C. **Google Cloud Storage** (to view our stored images)
 Gmail: officialbluepenguin@gmail.com
 Password: booyork70
 Bucket Link:
https://console.cloud.google.com/storage/browser/blue_penguin;tab=objects?authuser=2&hl=en&project=astute-harmony-441702-v4&prefix=&forceOnObjectsSortingFiltering=false

- D. **Figma Design:**
<https://www.figma.com/board/LrzOcF14DyZi7r03LPoATk/BluePenguin-Flow-Chart?node-id=0-1&t=7ixMAzXc6MI7GPyX-1>

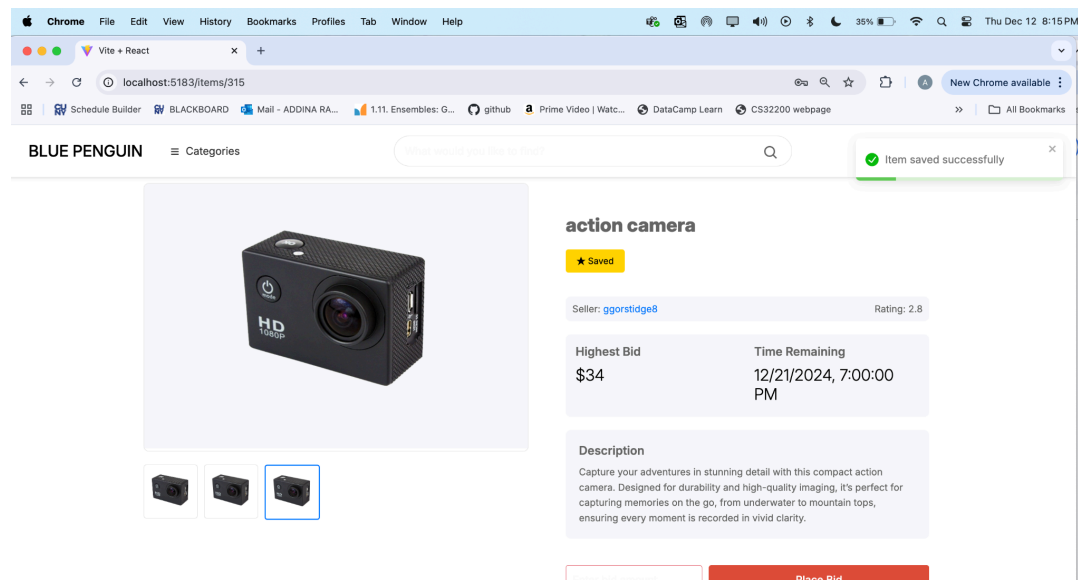
Blue Penguin – E-bidding Application	Version: 3.1
Software Requirement Specification	Date: 12/12/24
Third Draft	

5. Remarks

A. Database

We would like to clarify that our project is completely connected and synchronized with the database; all of our “dummy” items are also visible on our database, and are interactable as well; interacting with items on frontend does indeed update in our database. We used Supabase, a Postgresql-based Database Management System, which is a free alternative to Firebase.

Here is an example of the connection between our website and our database. Below is a screenshot of a “save” action occurring, which, as you can see in the upper right-hand corner of the entire screen, took place at 8:15 PM. In addition, the message pop-up shows that the item has been saved successfully at that moment. 8:15 PM EST is roughly 1:15 AM UTC of the next day (5 hour difference).



As you can see below, in our database’s table editor, this new save has been successfully added, with the automatically generated timestamp being 1:15 AM UTC. This specific table is the `api_saves` table, and there are foreign key relations to the `item_id` and `profile_id`.

Table Editor

schema: public

+ New table

Search tables...

api_note

api_parcel

api_paypaldetails

api_profile

api_quitrequest

api_rating

api_report

api_save

api_shippingaddress

BluePenguin

Free

/

BluePenguin2

Connect

Enable branching

Feedback

Filter

Sort

Insert

RLS disabled

role postgres

Realtime off

API Docs

	id int8	time_saved timestampz	ite... i...	profil... i...	+
	4	2024-12-08 21:29:50.426+00	358	54	
	6	2024-12-09 00:07:57.851+00	327	54	
	7	2024-12-09 03:47:43.897+00	462	54	
	11	2024-12-12 17:33:09.135+00	236	57	
	16	2024-12-12 20:34:02.088+00	466	54	
	17	2024-12-12 20:50:01.208+00	246	54	
	22	2024-12-13 01:15:31.682+00	315	54	

In the table `api_item`, `item_id` 315 is correctly attributed to the “action camera” item.

Table Editor

schema: public

+ New table

Search tables...

api_account

api_bid

api_carddetails

api_collection

api_comment

api_dislike

api_item

api_like

BluePenguin

Free

BluePenguin2

Connect

Enable branching

Filtered by 1 rule

Sort

Insert

RLS disabled

role postgres

Realtime off

API Docs


	id int8	image_urls jsonb	title text	description text	highest_bid numeric	deadline
	315	["https://storage.googleapis.com/blue_penguin/items/action_camera.jpg"]	action camera	Capture your adventures in stunning data	34.00	2024-12-13 01:15:31.682+00

Below, you can see that this update on the database has been reflected within the frontend as well, since you can view the item you have just saved under the “Saved” category.


BLUE PENGUIN
Categories
My Account

Pending
Saved
Next Actions
Awaiting Arrival


Saved Items




action camera
Current Price: \$34
Seller: ggorstidge8
Saved on: 12/12/2024, 8:15:31 PM
Ends: 12/21/2024, 7:00:00 PM



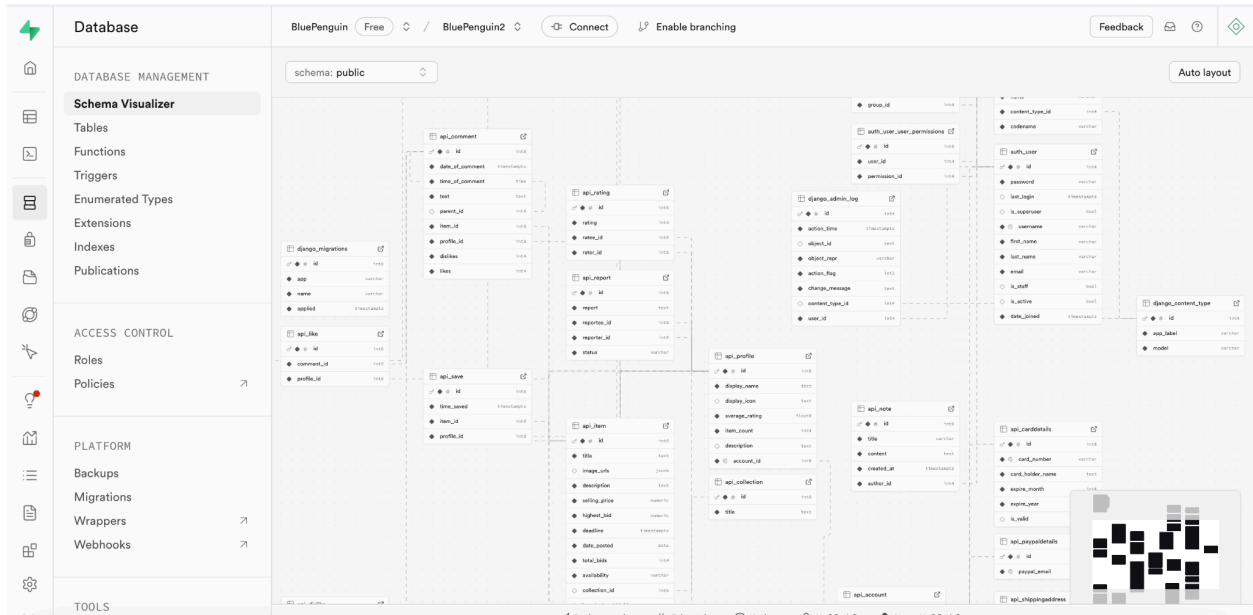
Skipper Plushie
Current Price: \$7
Seller: lily
Saved on: 12/12/2024, 3:34:02 PM
Ends: 12/12/2024, 4:37:00 PM



nintendo switch
Current Price: \$163
Seller: jroned
Saved on: 12/18/2024, 7:07:57 PM
Ends: 12/18/2024, 7:00:00 PM



Below is a class diagram of our entire schema as well, which are the same tables, attributes, and foreign keys defined in `/backend/api/models.py`.



In `/backend/backend/settings.py`, we have correctly configured our database to our Django backend using credentials provided by Supabase and located in our `.env` file in the backend directory.

```
settings.py × tasks.py middleware.py .gitignore backend .env ./ .env backend celery.py __init__.py ...  
backend > backend > settings.py > ...  
117 ]  
118  
119 WSGI_APPLICATION = 'backend.wsgi.application'  
120  
121  
122 # Database  
123 # https://docs.djangoproject.com/en/5.1/ref/settings/#databases  
124  
125 DATABASES = {  
126     'default': {  
127         'ENGINE': 'django.db.backends.postgresql',  
128         'NAME': os.getenv('DB_NAME'),  
129         'USER': os.getenv('DB_USER'),  
130         'PASSWORD': os.getenv('DB_PASSWORD'),  
131         'HOST': os.getenv('DB_HOST'),  
132         'PORT': os.getenv('DB_PORT'),  
133     }  
134 }  
135  
136
```

Below is Supabase's authentication table, which logs registered users. At the very top, you can see that the email we registered with in our demo has been added to the database. In addition, we have “dummy” users and dummy emails registered as well; so, they are indeed part of our database.

UID	Display name	Email	Phone	Providers	Provider type
2e769ba8-a81b-4b9c-a677-64b10c9b02a6	-	samiul@gmail.com	-	Email	-
3520449b-6177-4d48-9c62-dac2d0d0b828	-	prometheus@gmail.com	-	Email	-
7926be8d-55c9-4114-8a41-53d2cc244966	-	duality@gmail.com	-	Email	-
00777b06-62c9-4b2a-be47-dcad7420e0ec	-	samiul.saimon@macaulay.cuny.edu	-	Email	-
7e2138f8-91db-4b53-b818-d76e790c4c91	-	idamanallgt@gmail.com	-	Email	-
22cd8a4f-635a-44ea-9ee2-014d91dc864d	-	worthyforthe62@gmail.com	-	Email	-
e119a465-db77-4643-980d-9a762e877668	-	bdaffornef@weebly.com	-	Email	-
2c8c81df-0eb8-48a1-b538-65b5c28ee628	-	elockyear4@rediff.com	-	Email	-
6bc80f77-2197-4414-9a38-b56ef9097df5	-	sshensfisch1d@eepurl.com	-	Email	-
b017966b-d332-446c-a8e1-24c60eb7a181	-	eabbattic@cnbc.com	-	Email	-
92bbb987-b6c3-4f1e-8cb4-f3acad0ddb4d	-	lilenoirefb@netscape.com	-	Email	-

The issue that we encountered was not with the database itself, but the authentication system that we used with Supabase's API. This authentication system would not allow us to log in with these dummy emails, even though they have been “authenticated” under Supabase's authentication table. However, we are able to log in with emails and user data that have been manually registered directly through the website. As you have seen from the previous screenshots, our database is indeed properly connected with our website, and dummy data can generally be interacted with properly.

B. Background Tasks/Logic

i. Suspensions

Since suspensions hinge on ratings, we have implemented suspension logic in our “Rating” model. Below is a screenshot of the implementation on `/backend/api/models.py`.

Under the “save” method (meaning this occurs right after a Rating is added), the person who was rated (in this case, “ratee”) is fetched and we use Django ORM to fetch all ratings where this user was rated. If there are at least 3 ratings, we calculate the average rating using an aggregate method. We then implemented the following conditional logic:

- First check if the average rating is under 2 or above 4.
- If the status of the user is VIP, then simply change their status to “User.”
- If the status is not VIP, then suspend the account by setting `is_suspended` to “True.”

- Their item listings will be subsequently deleted, the number of suspension strikes will be incremented, and these changes will be saved to the database using the save() method.
- If the number of suspension strikes has reached 3, then the user will be permanently deleted from the website and database.

In our demonstration, we have shown how users who have been suspended can be unsuspended by Superusers in the admin account.

```

226
227 class Rating(models.Model):
228     rater = models.ForeignKey(Profile, on_delete=models.CASCADE, related_name='ratings_given')
229     ratee = models.ForeignKey(Profile, on_delete=models.CASCADE, related_name='ratings_received')
230     rating = models.PositiveIntegerField(validators=[MinValueValidator(1), MaxValueValidator(5)], default=0)
231
232     def save(self, *args, **kwargs):
233         super().save(*args, **kwargs)
234
235         ratee = self.ratee
236         ratee_user = ratee.account.user
237         ratings = Rating.objects.filter(ratee=ratee)
238
239         if ratings.count() >= 3:
240             avg_rating = ratings.aggregate(Avg('rating'))['rating_avg']
241             is_vip = False
242             if avg_rating < 2 or avg_rating > 4:
243                 if ratee.account.status == STATUS_VIP:
244                     ratee.account.status = STATUS_USER
245                     is_vip = True
246                 else:
247                     ratee.account.is_suspended = True
248
249                 ratee.account.suspension_strikes += 1
250
251                 ratee_items = Item.objects.filter(profile=ratee)
252                 ratee_items.delete()
253                 ratee.account.save()
254
255                 buyers = Transaction.objects.filter(seller=ratee).select_related('buyer_user').values_list('buyer_user', flat=True).distinct()
256                 for buyer in buyers:
257                     items = Transaction.objects.filter(seller=ratee, buyer_user=buyer).select_related('bid_item').values_list('bid_item_title', flat=True)
258                     item_titles = list(items)
259                     EmailNotifications.notify_items_deleted(buyer, item_titles)
260
261                 reason = "Average rating has reached less than 2 — Too mean." if avg_rating < 2 else "Average rating has reached greater than 4 — Too generous."
262                 EmailNotifications.notify_account_suspended(ratee_user, reason, is_vip)
263
264                 if ratee.account.suspension_strikes >= 3:
265                     EmailNotifications.notify_account_permanently_suspended(ratee_user)
266                     ratee.account.delete_account_user_profile()
267

```

ii. VIP system

For the VIP system, we have also defined model methods and implemented them in views methods. Below is a screenshot of the methods under the “Account” model. The “Account” model (which is the api_account table in the database) has an attribute called “Status” which we will play around with when looking at VIPs.

The first one is the become_VIP method which checks if a user has “User” status, then changes the status to VIP, which is then saved to the database using the save() method.

The second is the revoke_VIP method which checks if a user has “VIP” status, then changes the status back to “User,” and these changes are subsequently saved to the database.

These two methods are then called in “check_vip_eligibility.” This method checks how many transactions they have completed as both a buyer and a seller, checks if they have any reports/complaints from other users, and checks if the balance is sufficient enough. If the user has status “User” and has the right requirements, then the become_VIP method is called. Else, if the user has status “VIP”, and has reports/complaints or an

insufficient balance (number of transactions cannot decrease so we did not check that), then their VIP status is revoked.

In the method `get_VIP_discount`, 10% of the actual price is added back to a user's balance IF they have VIP status, which performs the discount.

```
backend > api > models.py > Rating > save
57 class Account(models.Model):
69     def delete_account_user_profile(self):
70
71         self.delete()
72
73         user.delete()
74
75         return True
76     except:
77         return False
78
79     def become_VIP(self):
80         if self.status == STATUS_USER:
81             self.status = STATUS_VIP
82             self.save()
83
84     def revoke_VIP(self):
85         if self.status == STATUS_VIP:
86             self.status = STATUS_USER
87             self.save()
88
89     def check_vip_eligibility(self):
90         transaction_count = Transaction.objects.filter(
91             Q(seller=self) | Q(buyer=self)
92         ).count()
93
94         has_reports = Report.objects.filter(reportee=self.profile).exists()
95         balance_sufficient = self.balance > 5000
96
97         if self.status == STATUS_USER and transaction_count > 5 and not has_reports and balance_sufficient:
98             self.become_VIP()
99             EmailNotifications.notify_VIP_status_earned(self.user)
100         elif self.status == STATUS_VIP:
101             if has_reports or not balance_sufficient:
102                 self.revoke_VIP()
103                 EmailNotifications.notify_VIP_status_revoked(self.user)
104
105     def get_VIP_discount(self, actual_price):
106         if self.status == STATUS_VIP:
107             # get a 10% discount ; means that you get 10% of the price added back to balance
108             new_price = actual_price * 0.10
109             self.balance += new_price
110             self.save()
```

These methods are subsequently called in a viewset action called `choose_winner` in `/backend/api/views.py`. This action is linked to the API endpoint `'api/items/{pk}/choose-winner,'` where `'pk'` is the primary key of the item object of the database.

Below is a screenshot of the action. As you can see in this action, after the `"complete_transaction"` function (which adds money to the seller's balance and subtracts money from the buyer's balance, as can be seen in its definition in `/backend/api/utils.py`), the method `get_VIP_discount` is called for the buyer, which is supposed to give 10% of the original amount back IF the buyer is a VIP. Afterwards, a transaction object is created and added to the database. Since there has just been a transaction, this means that the account balance of both the buyer and the seller has been updated, as well as their number of transactions; this means that they can potentially have their VIP status earned

or revoked. For this reason, `check_vip_eligibility` was called on both the buyer and the seller.

```
backend > api > views.py > AccountViewSet > add_to_balance
837 class ItemViewSet(viewsets.ModelViewSet):
1149 @action(detail=True, methods=['post'], permission_classes=[AllowAny, IsOwner], url_path='choose-winner')
1150 def choose_winner(self, request, pk=None):
1151     try:
1152         item = self.get_object()
1153         try:
1154             bid_id = request.data.get('id')
1155             winning_bid = Bid.objects.get(id=bid_id, item=item)
1156         except Bid.DoesNotExist:
1157             return Response(
1158                 {
1159                     "error": "Bid not found."
1160                 }, status=status.HTTP_404_NOT_FOUND
1161             )
1162
1163         seller_account = item.profile.account
1164         buyer_account = winning_bid.profile.account
1165
1166         complete_transaction(
1167             seller=seller_account,
1168             buyer=buyer_account,
1169             amount=winning_bid.bid_price
1170         )
1171
1172         buyer_account.get_VIP_discount()
1173         buyer_account.update_points(winning_bid.bid_price)
1174
1175         transaction = Transaction.objects.create(
1176             seller=seller_account,
1177             buyer=buyer_account,
1178             bid=winning_bid,
1179         )
1180
1181         seller_account.check_vip_eligibility()
1182         buyer_account.check_vip_eligibility()
1183
1184         item.availability = SOLD_CHOICE
1185         item.winning_bid = winning_bid
1186         item.save()
1187
1188         winning_bid.winner_status = WINNING_PENDING_CHOICE
1189         winning_bid.save()
1190
```

Similarly, in the Report model (found in `/backend/api/models.py`), once a report object is created, the method `check_vip_eligibility` is also called on the person who was reported (the “reportee.”)

```
class Report(models.Model):
    reporter = models.ForeignKey(Profile, on_delete=models.CASCADE, related_name='reports_given')
    reportee = models.ForeignKey(Profile, on_delete=models.CASCADE, related_name='reports_received')
    report = models.TextField(max_length=1000)
    status = models.CharField(max_length=1, choices=REQUEST_STATUS_CHOICES, default=REQUEST_PENDING_CHOICE)

    def save(self, *args, **kwargs):
        super().save(*args, **kwargs)

        reportee = self.reportee
        reportee.account.check_vip_eligibility()

        reporter_user = self.reporter.account.user
        EmailNotifications.notify_report_received(reporter_user)
```