

Commerce Bank Project

Coy Kwan - Project Manager

Cori Mroz - Fullstack

Anna Johnson - Database/Backend

Feng Zheng - Fullstack/testing

Daylan Quinn - Front End

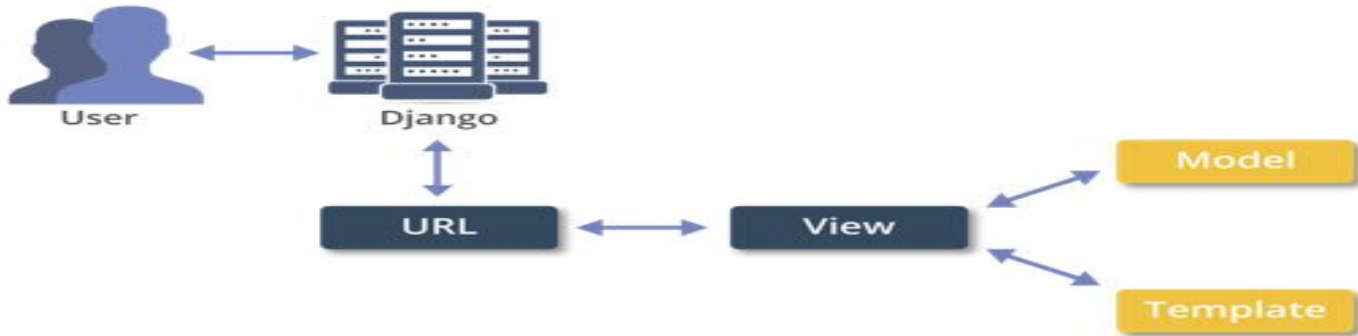
Tech Stack

- Python
- Django
- HTML
- CSS
- Bootstrap
- MySQL

Django Recap

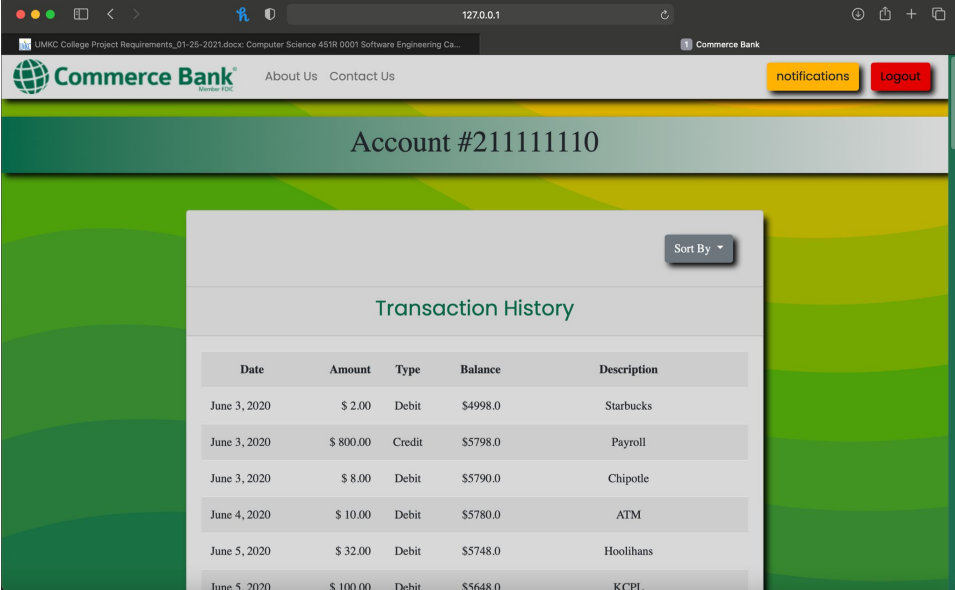
- Model -> View -> Template hierarchy
 - Models for database abstraction, views for middleware/api, templates for front end
- Provides admin tool for quick data visualization, manipulation for manual testing and added functionality.

Model View Template



Dashboard

- General transaction information
 - Ability to retroactively sort transactions by any of the notification rules
 - Access to notification settings



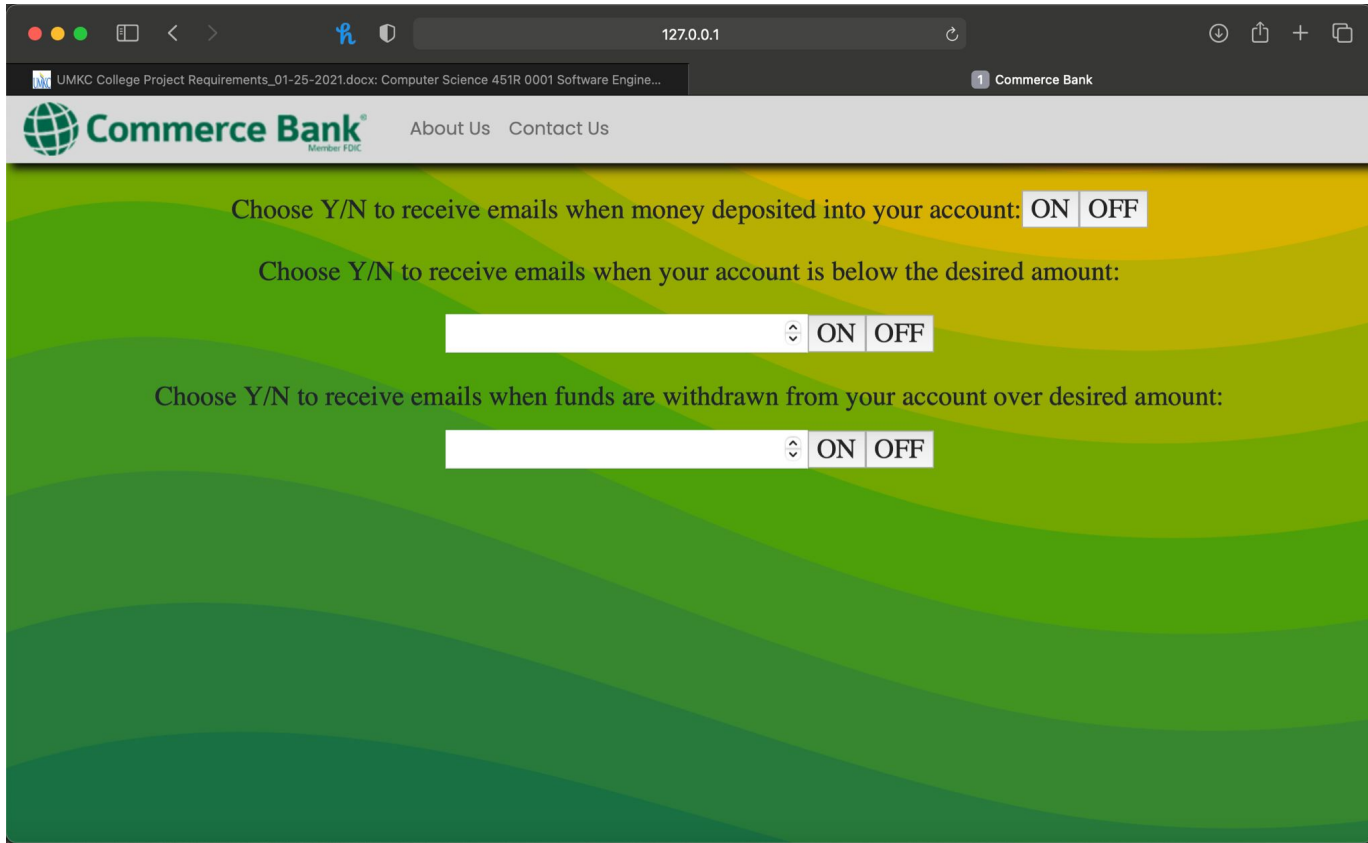
The screenshot shows a web browser window displaying the Commerce Bank dashboard. The browser's address bar shows the URL "127.0.0.1". The page header includes the Commerce Bank logo, navigation links for "About Us" and "Contact Us", and buttons for "notifications" and "Logout". The main heading is "Account #21111110". Below this, a "Transaction History" modal is open, featuring a "Sort By" dropdown menu. The table lists transactions from June 3, 2020, to June 5, 2020, with columns for Date, Amount, Type, Balance, and Description.

Date	Amount	Type	Balance	Description
June 3, 2020	\$ 2.00	Debit	\$4998.0	Starbucks
June 3, 2020	\$ 800.00	Credit	\$5798.0	Payroll
June 3, 2020	\$ 8.00	Debit	\$5790.0	Chipotle
June 4, 2020	\$ 10.00	Debit	\$5780.0	ATM
June 5, 2020	\$ 32.00	Debit	\$5748.0	Hoolihans
June 5, 2020	\$ 100.00	Debit	\$5648.0	KCPL

Notification Center


- Three basic rules
 - When money is deposited
 - When account drops below a threshold set by a user
 - When a withdrawal exceeds a threshold
- Notifications are set via email
 - Emails currently only “sent” locally however can be provided with a SMTP port for true functionality (All the decent email servers we found cost money)
 - EMAIL_BACKEND =
 - 'django.core.mail.backends.smtp.EmailBackend'
 - EMAIL_HOST = 'smtp.gmail.com'
 - EMAIL_USE_TLS = True
 - EMAIL_PORT = 12345
 - EMAIL_HOST_USER = 'your_account@gmail.com'
 - EMAIL_HOST_PASSWORD = 'your account's password'

Notification Center



127.0.0.1

UMKC College Project Requirements_01-25-2021.docx: Computer Science 451R 0001 Software Engine... Commerce Bank

 **Commerce Bank**
Member FDIC

[About Us](#) [Contact Us](#)

Choose Y/N to receive emails when money deposited into your account:

Choose Y/N to receive emails when your account is below the desired amount:

Choose Y/N to receive emails when funds are withdrawn from your account over desired amount:

Database

- Utilizing a MySQL database. Tables for:

```
class Transactions(models.Model):
    transaction_id = models.AutoField(primary_key=True)
    account_id = models.IntegerField(blank=True, null=True)
    processing_date = models.DateField(blank=True, null=True)
    balance = models.FloatField(blank=True, null=True)
    transaction_type = models.CharField(max_length=10, blank=True, null=True)
    amount = models.FloatField(blank=True, null=True)
    descr = models.CharField(max_length=50, blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'transactions'
```

```
class WithdrawalNotif(models.Model):
    account_id = models.IntegerField(primary_key=True)
    is_true = models.CharField(max_length=1)
    amount = models.FloatField(blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'withdrawal_notif'
```

```
class DepositNotif(models.Model):
    account_id = models.IntegerField(primary_key=True)
    is_true = models.CharField(max_length=1)

    class Meta:
        managed = False
        db_table = 'deposit_notif'
```

```
class BalanceNotif(models.Model):
    account_id = models.IntegerField(primary_key=True)
    is_true = models.CharField(max_length=1)
    amount = models.FloatField(blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'balance_notif'
```

Database and Middleware

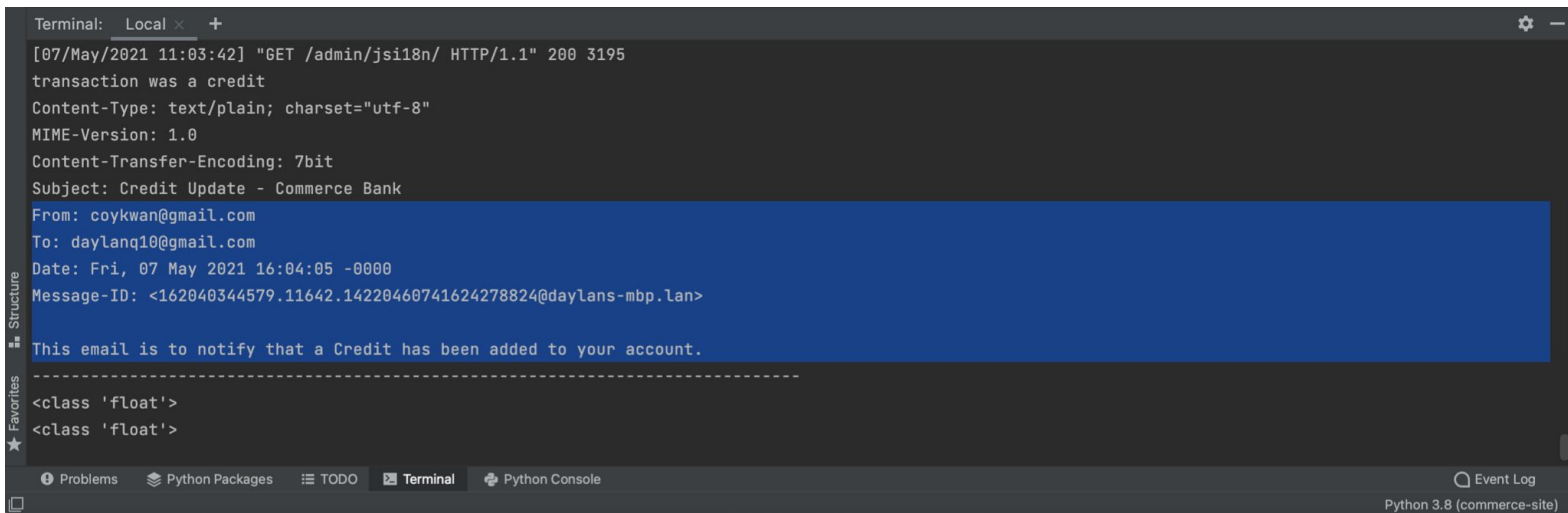
- Data represented in DDL like models in Django's MVT structure
- Views handle the logic allowing front end, templates, to talk to database
 - HTTP GET and POST requests
 - Rapidly speeds up development times

Security

- Django natively protects against
 - Cross site scripting
 - Request forgery using built in secret keys for HTTP requests
 - MVT protects against SQL injection to an extent
- Measures we have taken
 - Completely parameterized queries - no raw sql
 - Inputs heavily restricted

Stretch Goals

- Github used for source control
 - Pull requests performed to combine all of the different pieces into the main branch
 - Code review done throughout design process
- Notifications sent via email

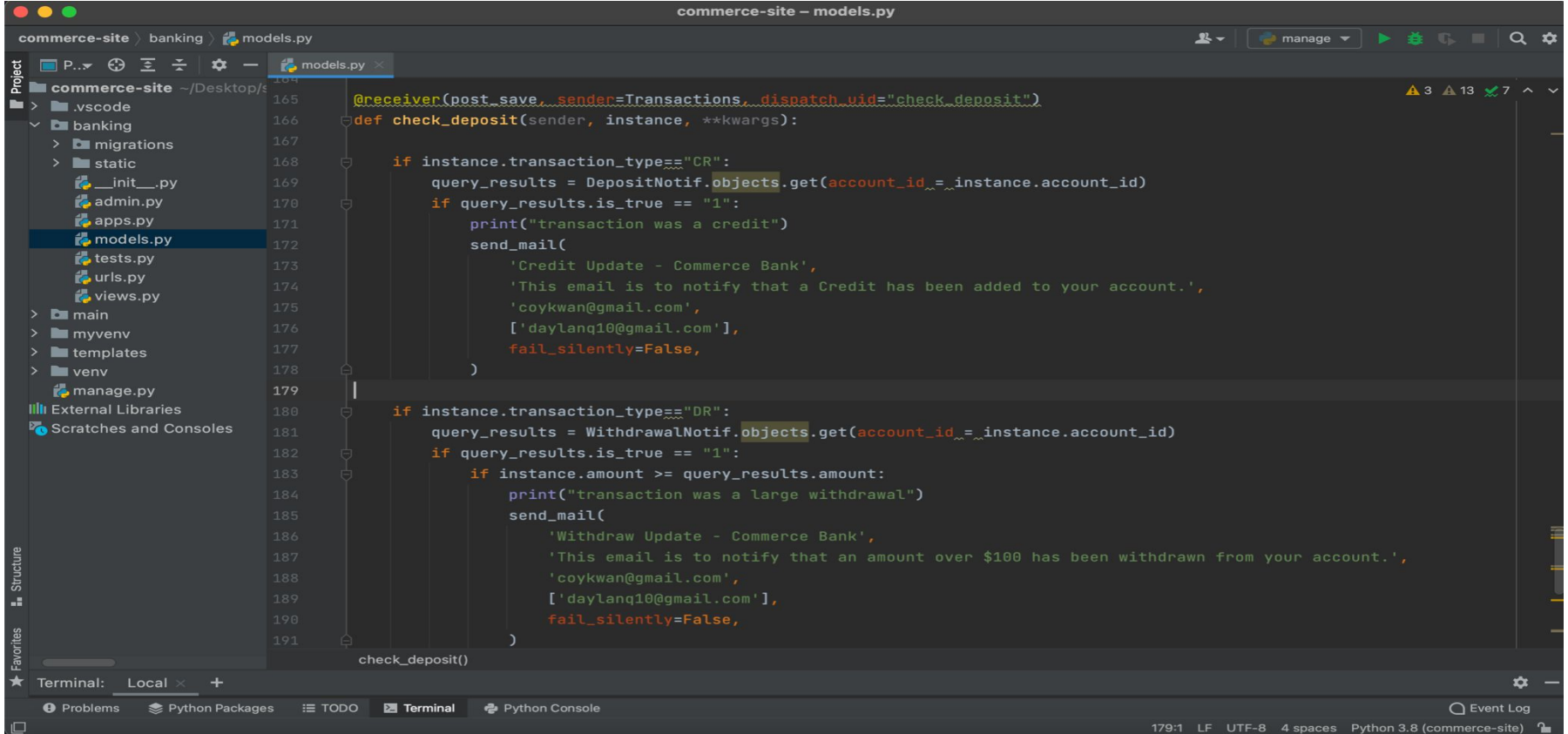


The screenshot shows a terminal window with a dark background. The title bar reads "Terminal: Local x +". The terminal output shows an HTTP GET request and an email notification. The email body is highlighted in blue. The bottom of the terminal shows a sidebar with "Structure", "Favorites", "Problems", "Python Packages", "TODO", "Terminal", and "Python Console". The status bar at the bottom right indicates "Python 3.8 (commerce-site)".

```
Terminal: Local x +
[07/May/2021 11:03:42] "GET /admin/jsi18n/ HTTP/1.1" 200 3195
transaction was a credit
Content-Type: text/plain; charset="utf-8"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit
Subject: Credit Update - Commerce Bank
From: coykwan@gmail.com
To: daylanq10@gmail.com
Date: Fri, 07 May 2021 16:04:05 -0000
Message-ID: <162040344579.11642.14220460741624278824@daylans-mbp.lan>

This email is to notify that a Credit has been added to your account.
-----
<class 'float'>
<class 'float'>
```

Email Notification Code



The screenshot shows a code editor window titled "commerce-site - models.py". The left sidebar displays a project structure for "commerce-site" located at ~/Desktop/. The structure includes folders like .vscode, banking, migrations, static, main, myenv, templates, and venv, along with files like __init__.py, admin.py, apps.py, models.py, tests.py, urls.py, and views.py. The "models.py" file is selected and open in the editor.

The code in "models.py" defines a Django signal receiver for the "post_save" signal of the "Transactions" model. The receiver is named "check_deposit" and is decorated with "@receiver(post_save, sender=Transactions, dispatch_uid='check_deposit')". It handles two transaction types: "CR" (Credit) and "DR" (Withdrawal).

```
165 @receiver(post_save, sender=Transactions, dispatch_uid="check_deposit")
166 def check_deposit(sender, instance, **kwargs):
167
168     if instance.transaction_type=="CR":
169         query_results = DepositNotif.objects.get(account_id=instance.account_id)
170         if query_results.is_true == "1":
171             print("transaction was a credit")
172             send_mail(
173                 'Credit Update - Commerce Bank',
174                 'This email is to notify that a Credit has been added to your account.',
175                 'coykwan@gmail.com',
176                 ['daylanq10@gmail.com'],
177                 fail_silently=False,
178             )
179
180     if instance.transaction_type=="DR":
181         query_results = WithdrawalNotif.objects.get(account_id=instance.account_id)
182         if query_results.is_true == "1":
183             if instance.amount >= query_results.amount:
184                 print("transaction was a large withdrawal")
185                 send_mail(
186                     'Withdraw Update - Commerce Bank',
187                     'This email is to notify that an amount over $100 has been withdrawn from your account.',
188                     'coykwan@gmail.com',
189                     ['daylanq10@gmail.com'],
190                     fail_silently=False,
191                 )
```

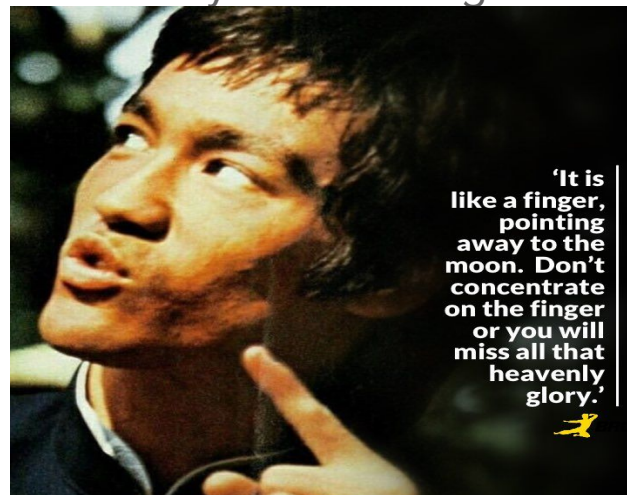
The bottom of the editor shows a terminal window with the text "Terminal: Local" and a plus sign to open a new terminal. The status bar at the very bottom indicates the current line is 179:1, the file is in LF format, UTF-8 encoding, 4 spaces for indentation, and the Python version is 3.8 (commerce-site).

Live Demo

<https://umkc.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=80a8921c-8c15-4fb3-8a3d-ad210117303d>

Risks and lessons learned

- HTTP requests are a hard concept for first and second time users even with built in view API
 - This should've been attacked much earlier
- Division of functionality would allow for faster and easier development
 - Initial planning of a hyper mobile friendly single page responsive design did not lend itself well to division of labor
- If you spend too much time thinking about a think you'll never get it done
 - Planning is great but action is better
 - Slow and steady wins the race



Questions?