BINUS UNIVERSITY
BINUS INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:	Surname	Given Names	Student ID Number
	Honasan	David	2101693933

Course Code	**: COMP6502**	**Course Name**	**: Introduction to Programming**
Class	**: L1BC**	**Name of Lecturer(s)**	**: 1. Jude Martinez** **2. Minaldi Loeis**
Major	**: Computer Science**		
Title of Assignment	**: Internet Banking**		
Type of Assignment	**: Final Project**		
Due Date	**: 7-11-2017**	**Submission Date**	**: 7-11-2017**

Plagiarism/Cheating

Binus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to Binus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student:

David Honasan

Internet Banking System

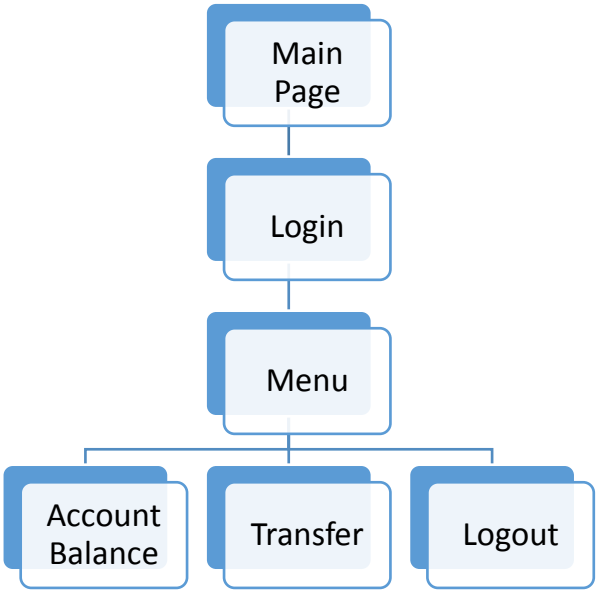
I. Description

The function of this program:

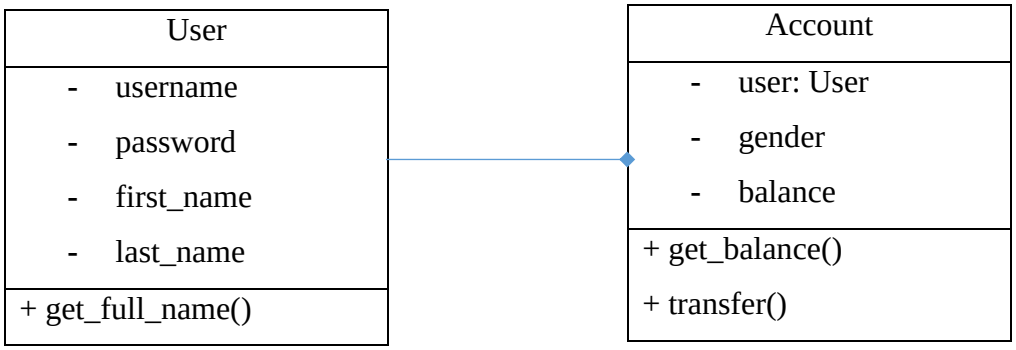
This program is a website of internet banking, where the user can login to their account and see their account balance and transfer his/her money to other account that is registered in the website.

II. Solution Design

1. **Plan**



2. **UML Diagram**



III. Explanation of the Program Code

1. internet_banking/settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # my apps
    'bank',
]
```

At INSTALLED_APPS, I type-in 'bank' as an installed apps, so Django could recognized the bank folder as an app.

```
# Login Url and redirect
LOGIN_URL = 'bank:login_form'    # if not login direct to this link
LOGIN_REDIRECT_URL = 'bank:menu'     # after login redirect to menu
```

I also created a new line 'LOGIN_URL' to login_form and 'LOGIN_REDIRECT_URL' to menu. So, it would redirect the user to the login form if they are accessing the menu page directly from the URL.

```
# Session expire
SESSION_COOKIE_AGE = 600 # Session expires after 600s
SESSION_EXPIRE_AT_BROWSER_CLOSE = True  # Session expired after browser closed
```

And I added 'SESSION_COOKIE_AGE' and set it to 600, meaning users that have been logged on will be logged out automatically after 600 seconds, which means 10 minutes.

'SESSION_EXPIRE_AT_BROWSER_CLOSE' = True meaning session will be expired or users that still logged on will be logged out if the browser is closed.

2. internet_banking/urls.py

```
from django.conf.urls import url, include
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', admin.site.urls),    # admin url
    url(r'^', include('bank.urls')),    # include the url file in bank folders
]
```

At the urls.py file, I imported 'include' so I can include my urls.py file in the bank folder.

3. bank/admin.py

```
from django.contrib import admin
from .models import Account

# Register your models here.
class AccountAdmin(admin.ModelAdmin):
```

```
    readonly_fields = ('balance',)  # make the balance read only in admin page

admin.site.register(Account, AccountAdmin)  # register the model in admin site
```

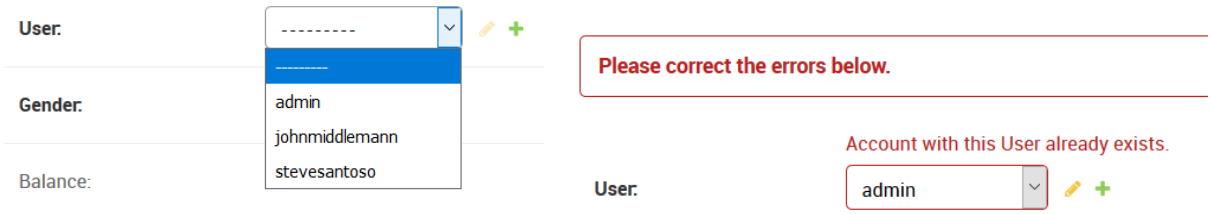
At the admin.py file, I imported class Account from the model files. And at the AccountAdmin class, I made the balance fields that will appear at the admin page is read only. So, it couldn't be changed directly from the page by the admin.

admin.site.register meaning register the model class into the admin site, so we can add, edit, or delete data easily.

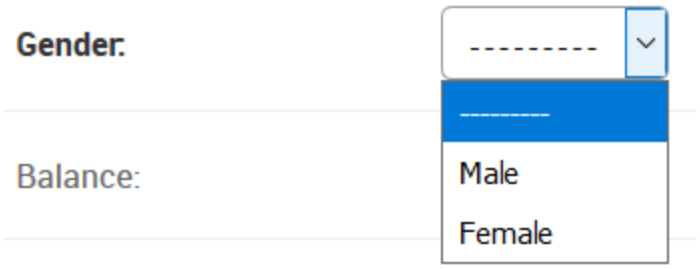
4. **bank/models.py**

```
from django.db import models


# Create your models here.
class Account(models.Model):
    user = models.OneToOneField('auth.User')   #link this field with Django User.
    GENDER = (
        ('M', 'Male'),
        ('F', 'Female'),
    )
    gender = models.CharField(max_length=1, choices=GENDER) # gender field
    balance = models.IntegerField(default=0)    # balance field

    def __str__(self):
        return self.user.get_full_name()    # return full name in admin site.
```

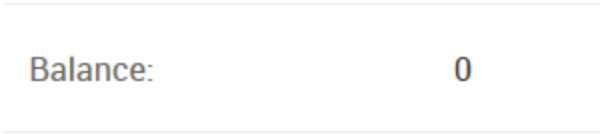
At the models.py file, I imported models as the class is going to be a table in the database file. And also, I create my model with the name Account. The user field mean that it will link uniquely (one-to-one relationship) this column with auth.User (Django User Authorization). So it will link all data from User to Account.



It will show that Account with this User already exists if the user already created an account.

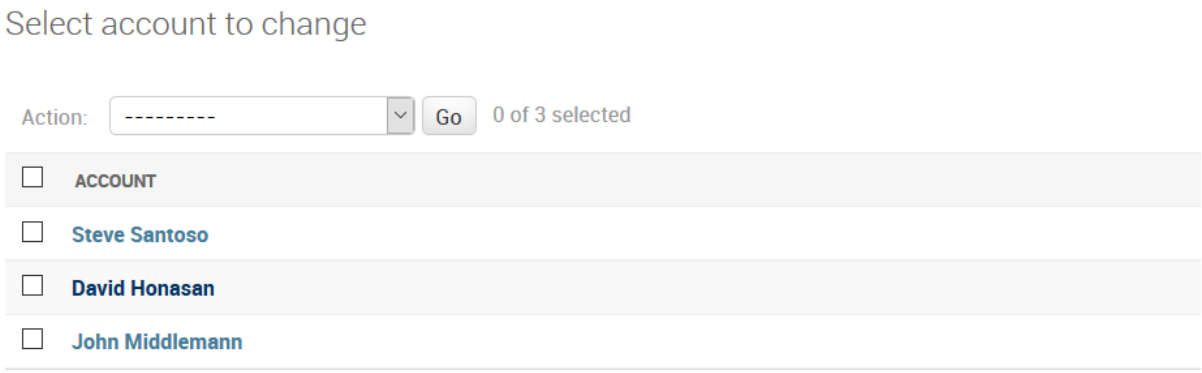


At the gender field, I created it with choices so you could choose your gender. Not a really useful account info actually.



And also a zero balance when creating a new account as the default value is 0 and it's a read only field.

The def __str__(self) is a function to show the Account name as a full name not an 'Account object' thingy. As shown at the picture below:



5. bank/urls.py

So, I actually create this new file because it wasn't generated by the 'startapp' command.

```
from django.conf.urls import url
from . import views

app_name = 'bank' #app name for namespace

urlpatterns = [
    url(r'^$', views.index, name='main'),    # main page
    url(r'^login/$', views.login_form, name='login_form'),  # login page
    url(r'^login/login$', views.login_action, name='login'),     # login action
    url(r'^logout/$', views.logout_action, name='logout'),  # logout action
    url(r'^menu/$', views.menu, name="menu"),    # menu
    url(r'^menu/balance/$', views.balance, name="balance"), # balance
    url(r'^menu/transfer/$', views.transfer_form, name="transfer_form"),# transfer
page
    url(r'^menu/transfer/transfer/$', views.transfer, name='transfer'), # transfer
action
]
```

First thing, I imported url function and views.py file. I also declared the app_name as 'bank' for namespace because I wanted to make dynamic not hardcoded link as dynamic link was preferred.

So, I wrote in the url patterns, this **r'…'** thingy is actually what the users request from their URL. It reads from the browser URL and configure what will it do (It will execute 'views.something'). I also wrote in the name='' for the namespace.

6. bank/views.py

This last part is actually what play its parts the most because mostly of the coding was right in here.

```
from django.shortcuts import render, redirect
from django.contrib.auth import authenticate, login, logout
from django.contrib.auth.decorators import \
    login_required
from django.contrib.auth.models import User
from django.contrib import messages  # importing message for alert if there's error
from django.core.exceptions import ObjectDoesNotExist  # importing raise error
from .models import Account  # importing account object
```

I imported all necessary modules. These are the explanations:

a. render: to render / show the html page
b. redirect: to redirect the user into other url
c. authenticate: to authenticate with the username and password and check if user exist or not
d. login: logging you to a session
e. logout: logging you out from the session.
f. login_required: for redirecting user to the login page if they're trying to access the page that needs login.
g. User: importing user object for something.
h. Message: to use Django's built-in module to show message (could be error message or info, etc).
i. ObjectDoesNotExist: to raise exception in try/except if object that trying to get doesn't exist.
j. Account: import account object for transaction or balance.

Some codes in the views.py are like only return render and basically do the same thing. Some example:

```
# main page
def index(request):
    return render(request, 'index.html')
```

This code is executed if user open the main page and Django will render the index/main page.

```
def login_form(request):
    if not request.user.is_authenticated:  # if not authenticated render login, else go to menu
        return render(request, 'account/login.html')
    else:
        return redirect('bank:menu')
```

If user isn't logged in render the login page, else redirect user's to menu.

```
def login_action(request):
    # get the username and password then authenticate it
    username = request.POST['username']
    password = request.POST['password']
    user = authenticate(request, username=username, password=password)
    if user is not None:  # if authenticate is successful then login and redirect to menu
        login(request, user)
        return redirect('bank:menu')
```

```
    else:  # else show error
        messages.error(request, 'Sorry, wrong username / password or either account
is not registered. ')
        return redirect('bank:login_form')
```

request.POST['name'] meaning that function is getting the value from the form with method POST in input name username and password. Then, it'll try to authenticate with function authenticate(request, username=username, password=password).

After that, if the user exist, then log him/her in and redirect user to menu. Else, show error message and show it below the form. This is the example of an error if occurred:

Error! Sorry, wrong username / password or either account is not registered.

```
def logout_action(request):
    logout(request)  # logout and redirect to main
    return redirect('bank:main')
```

This code is about logging out user if they click logout and will be redirected to the main page.

```
@login_required
def menu(request):
    name = request.user.get_full_name()  # show the full name
    return render(request, 'menu/menu.html', {'full_name': name})
```

This code have @login_required at the top of it. Meaning that, "You should've logged in first to see this page or else I'll redirect you to login page. " Something like that is Django what going to say or do.

The request.user.get_full_name() is to get the Full name of who's logged in now and render it in the page.

```
@login_required
def balance(request):
    acc = Account.objects.get(user_id=request.user.id)  # get the balance for the
user that logged on
    return render(request, 'menu/balance.html', {'acc': acc})
```

Same as the above, need to be logged in first. Then, get the object for it (by matching the user_id on the Account object and id at the User object.) and showing the balance in the page.

```
@login_required
def transfer_form(request):
    return render(request, 'menu/transfer.html')
```

Need to be logged in, and render the transfer form.

```
@login_required
def transfer(request):
    # Checking if user that receives the money exist.
    try:
        # Variables
        u_receive = request.POST['to']  # Username that will receive the money
        acc_sender = Account.objects.get(user_id=request.user.id)  # Getting the
sender account object
        username_receiver = User.objects.get(username=u_receive)  # Getting the
receiver object
        acc_receiver = Account.objects.get(user_id=username_receiver.id)  # Getting
the account of receiver object
        amt = int(request.POST['amount'])  # Getting the amount of transferring
```

```
        # Checking if transfer to yourself. because if you're transferring to
yourself there's money duplication bug.
        if u_receive != request.user.username:

            # Validates if amt less or equal than sender account balance.
            if amt <= acc_sender.balance and amt >= 10000:
                acc_sender.balance -= amt  # Subtracting your balance with the amount
of amt
                acc_receiver.balance += amt  # Adding balance with the amount of amt
                acc_sender.save()  # Saving balance that has been subtracted to DB.
                acc_receiver.save()  # Saving balance that has been added to DB.
                messages.success(request, 'You have transferred Rp. {} to {}. You now
only have Rp. {}'.format(
                    amt, u_receive, acc_sender.balance))
                return redirect('bank:transfer_form')
            else:
                messages.error(request,
                               'You don\'t have that much amount of balance or trying
to transfer below the minimum amount. You only have Rp. {}'.format(
                                   acc_sender.balance))
                return redirect('bank:transfer_form')
        else:
            messages.error(request, 'You can\'t transfer to yourself!')
            return redirect('bank:transfer_form')
    except ObjectDoesNotExist:
        messages.error(request,
                       'You can\'t transfer to \'{}\' because that user doesn\'t
exist.'.format(request.POST['to']))
        return redirect('bank:transfer_form')
```

So, this is actually the longest part of the code and it's functionality to transfer your money from your account to other designated account.

First with the try function, it'll go to except: if there are an error with ObjectDoesNotExist, which is going to be occurred in the username_receiver if there aren't valid receiver account.

Then, the variables u_receive to get the username of the receiver. acc_sender, getting the sender Account object. username_receiver, getting the User with that username. acc_receiver, getting the receiver Account object. amt, getting the amount from the form and convert it to integer because data that comes from forms are all in string.

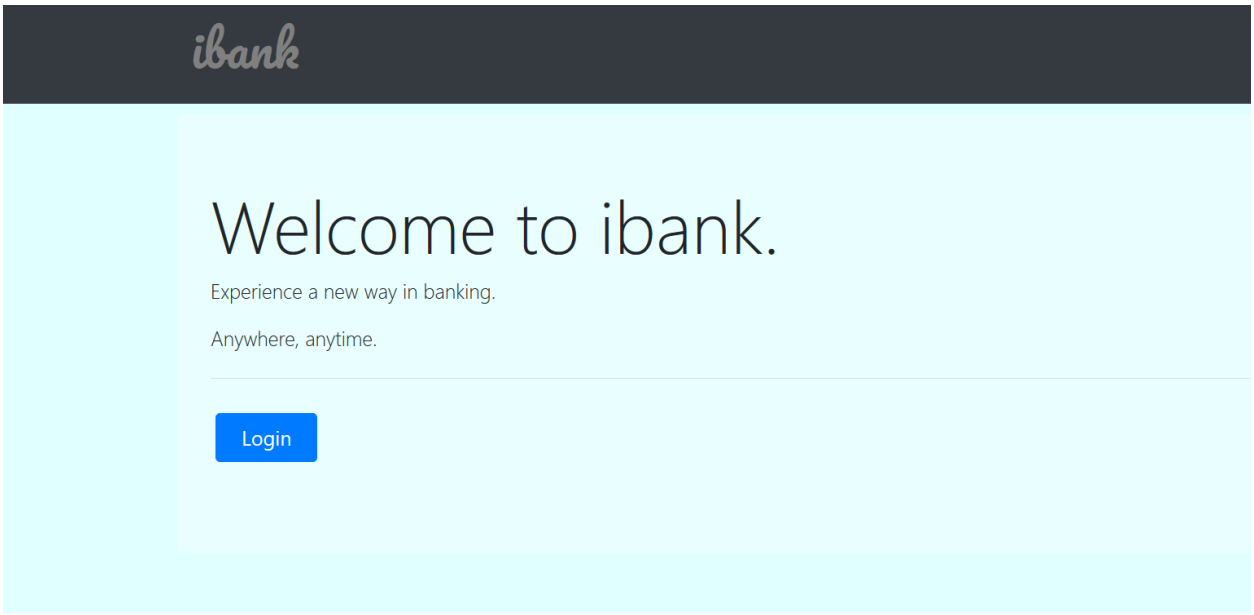
And also, the code will validates if you're going to send it to other person, not yourself, because there's a bug of money duplicating if you send it to yourself. It also validates if the amount of money you're going to transfer isn't bigger than your money in your account and the amount of transfer higher than minimum amount. Then it will reduce the money from the sender and add it to the receiver and save it to database. If there are errors, well they will show up at below the form like in the login page error.

```
{% if messages %}
    <div class="alert alert-danger">
        {% for message in messages %}
            <p{% if message.tags %} class="{{ message.tags }}"{% endif %}><strong>{{
message.tags }}! </strong>{{ message }}</p>
        {% endfor %}
    </div>
{% endif %}
```

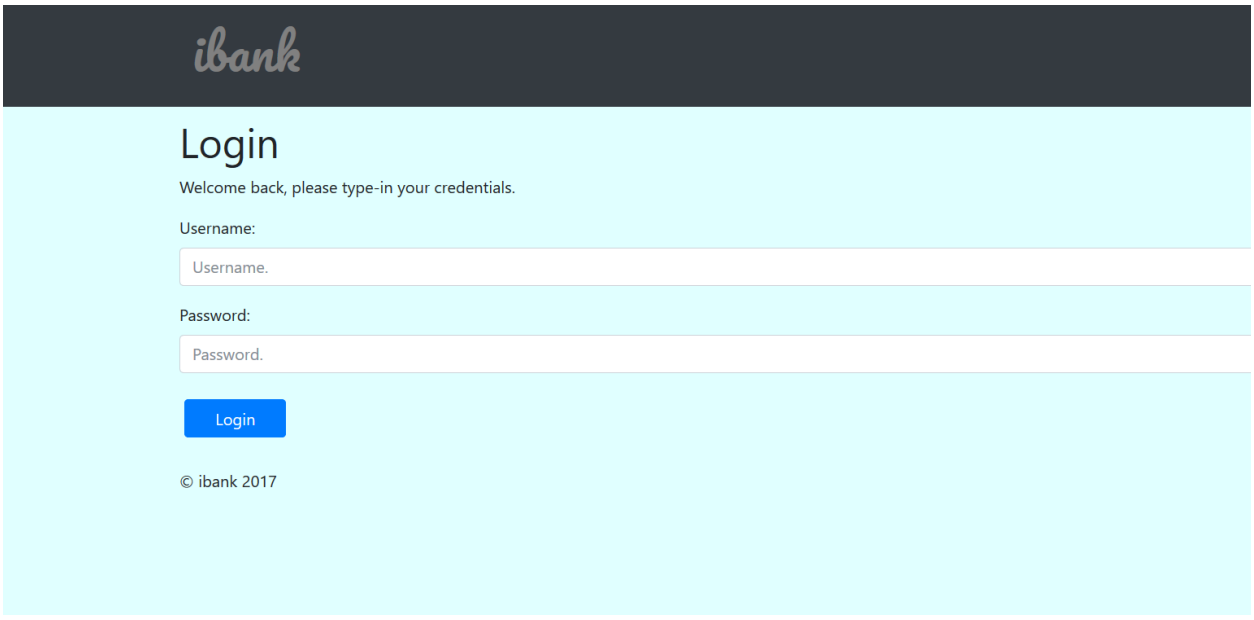
Example of how to showing message in html, in the .html page.

If there are messages, it will show message for every message there are available.

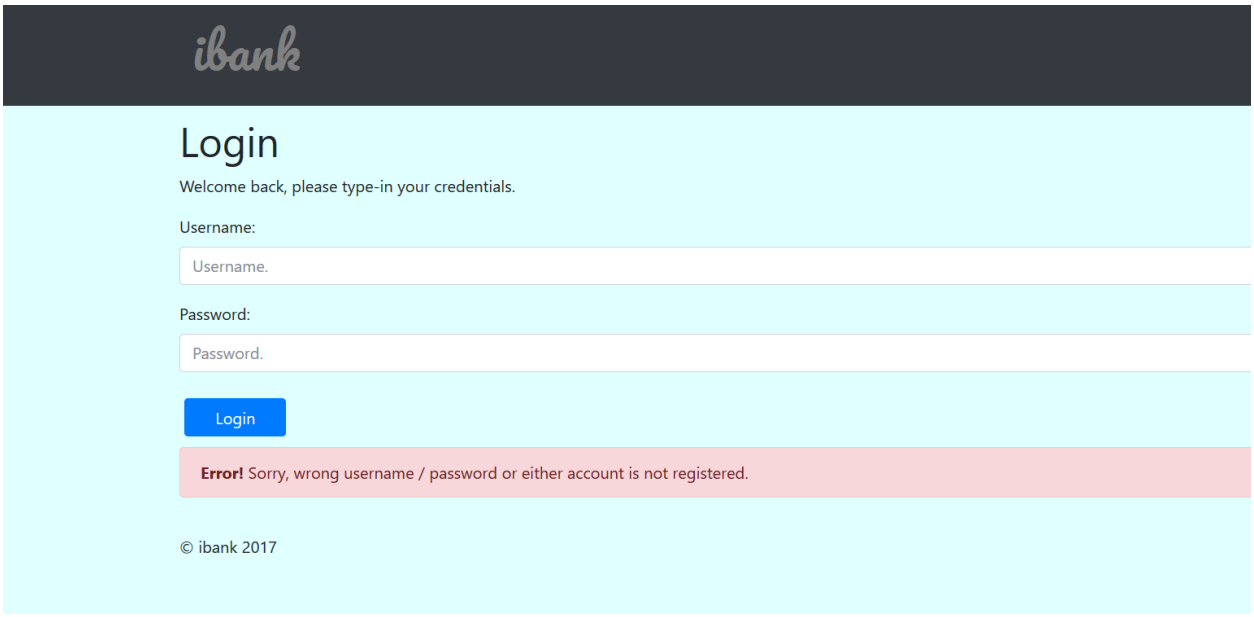
IV. Screenshots of Working Program



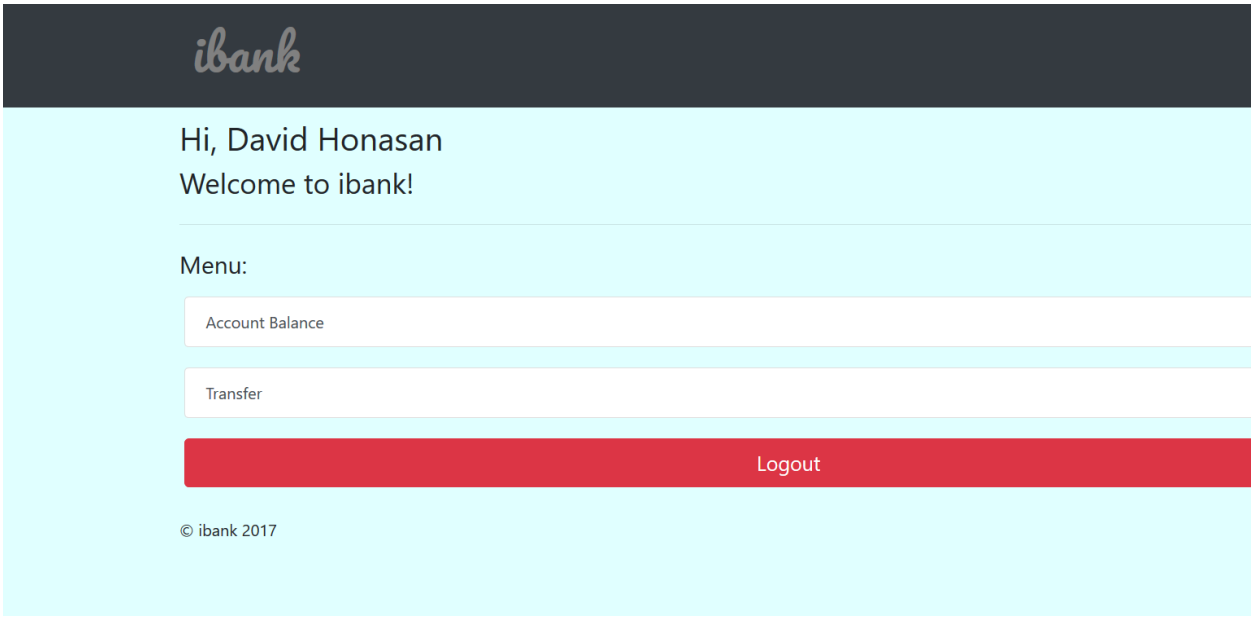
Main page.



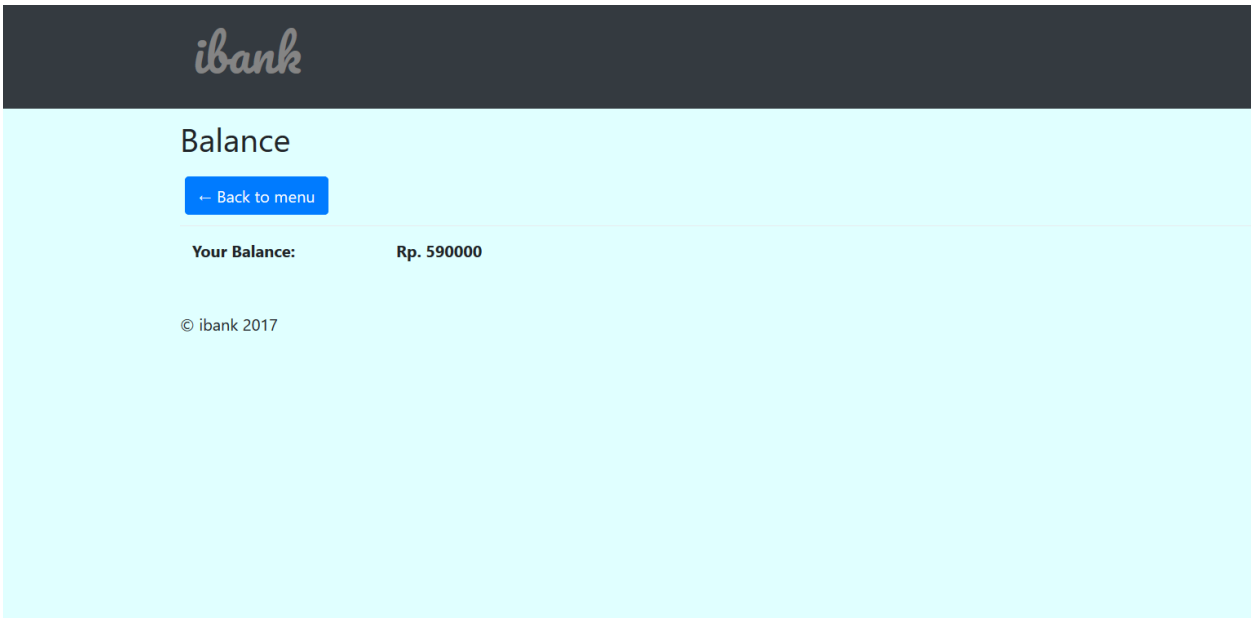
Login page.



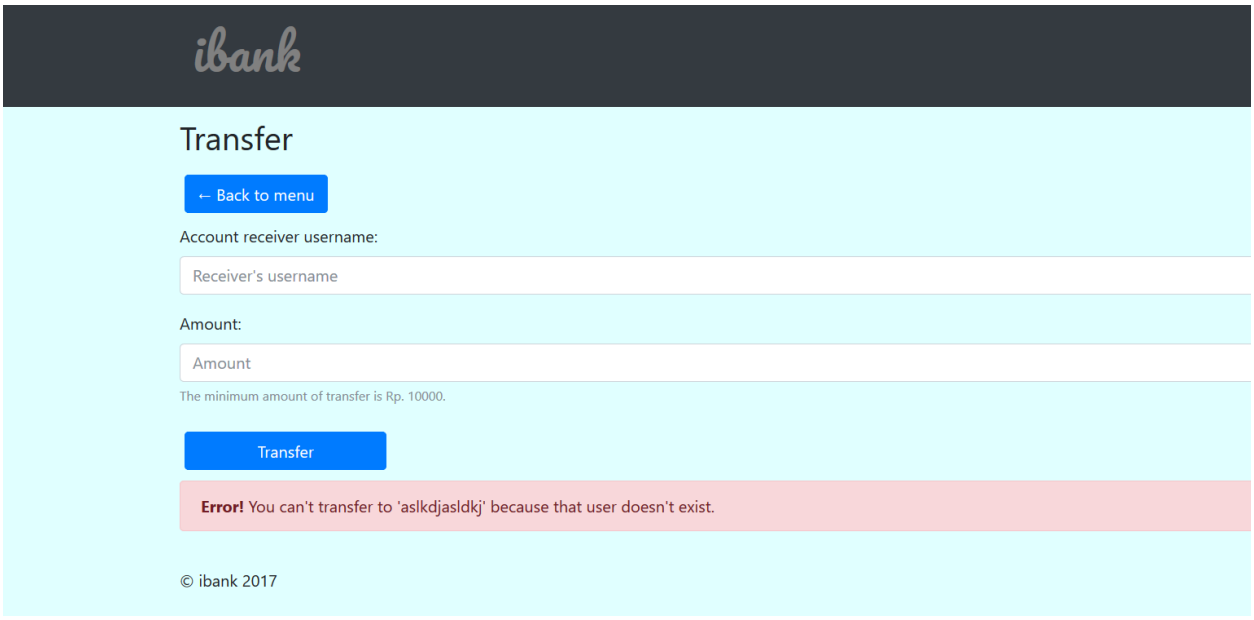
Error occurred in login page.



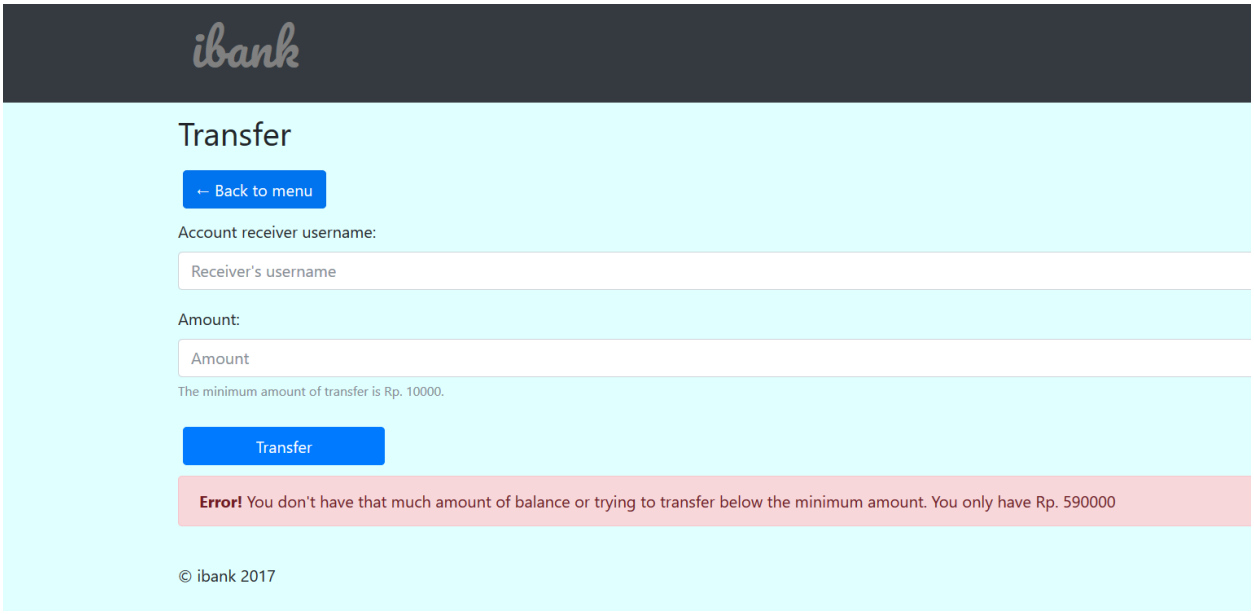
Menu, (notice that it also show my name or the account name.)



Balance page.



Transfer page, and error in transfer page occurred because I typed in random username receiver.



Error occurred because trying to send money more than your balance.

V. Source Code

I've pushed all of my code to GitHub:

https://github.com/davidhonasan/DavidHonasan_ITP2017_FinalProject

Sources

Django Documentation: https://docs.djangoproject.com/

Django Youtube tutorials by thenewboston:
https://www.youtube.com/playlist?list=PL6gx4Cwl9DGBlmzzFcLgDhKTTfNLfX1IK

Bootstrap CSS v4.0.0-beta.2: http://getbootstrap.com/

Some help by other class' TA: Aldi