PayDay

EC4 Final Year Project Presentation

By
Mihail Gaidau
Vincent Lloyd Yuson

What is PayDay ?

Payday

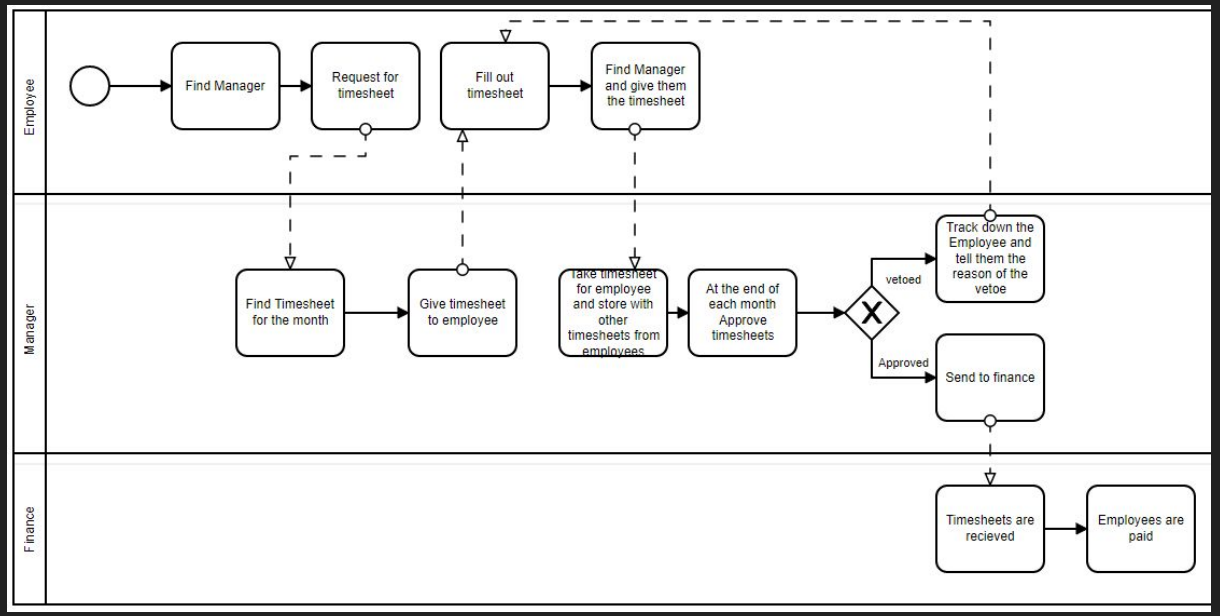
- PayDay is a time management system that digitizes timesheets

- The main idea of PayDay is to speed up and make the payroll process of any SME, be more efficient

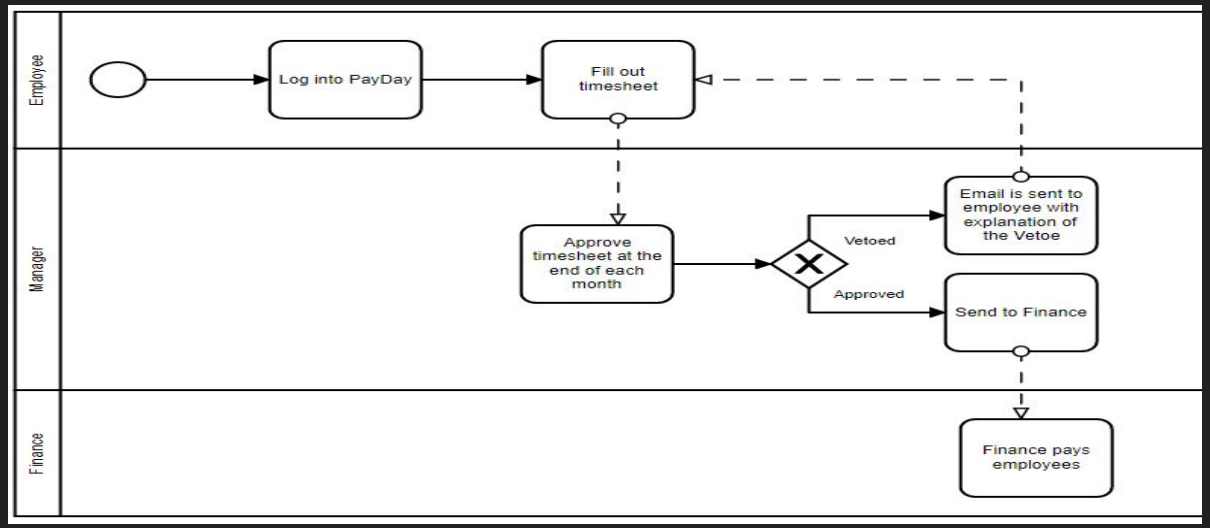
What we do?

- What we do is introduce digitized timesheets to your company.
- As a result, by introducing digitized timesheets, we also inadvertently alter your payroll process.
- The overall workflow becomes sped up and made way more efficient than conventional timesheet processes.
- By conventional, we mean payroll processes that are centered around paper timesheets

Paper/excel timesheet process



PayDay timesheet process



Technology used

Payday

- The technology used in the development of PayDay are as follows:
  - Python
  - Django
  - HTML
  - CSS
  - GIT
  - JavaScript
  - Bootstrap 4

Technology used

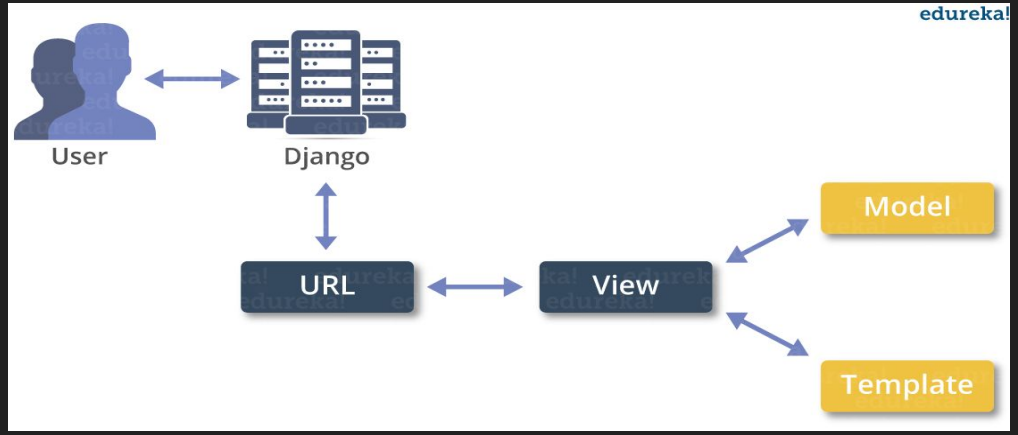
Payday

- Django and Python
  - Was used as the framework and what was used to build the skeleton of PayDay
- HTML, CSS, Bootstrap and JavaScript
  - These technologies was used to build the aesthetics of the website
- GIT
  - GIT was used do that both team members were able to work on the project remotely

Development with Django

Payday

Django Utilises the MVT model to output data onto the HTML template. Both model and Template talk to the view which is connected to the URL ultimately depending the what URL the user choose it will output the correlating data on the HTML template.



Template is where you write your HTML code along with where you call your CSS and JS files to style the UI.

Development with Django

Payday

These are the examples of the URL, View and Model of our Manager registration view.

We have also provided the Manager registration form on the left hand-side.

```
url(r'^emp_timesheet_history$', views.emp_timesheet_history_view, name='emp_timesheet_history'),
url(r'^man_timesheet_history$', views.man_timesheet_history_view, name='man_timesheet_history'),
```

```
def managerregistration_view(request):
    context = {}
    if request.POST:
        form = ManagerRegistrationForm(request.POST)
        if form.is_valid():
            form.save()
            email = form.cleaned_data.get('email')  # How to get data from a valid form
            raw_password = form.cleaned_data.get('password1')
            manager = authenticate(email=email, password=raw_password)  # how to create the account
            login(request, manager)
            return redirect('managerhome.html')
        else:
            context['managerregistration_form'] = form
    else:  # GET request
        form = ManagerRegistrationForm()
        context['managerregistration_form'] = form
        return render(request, 'managerregistration.html', context)
```

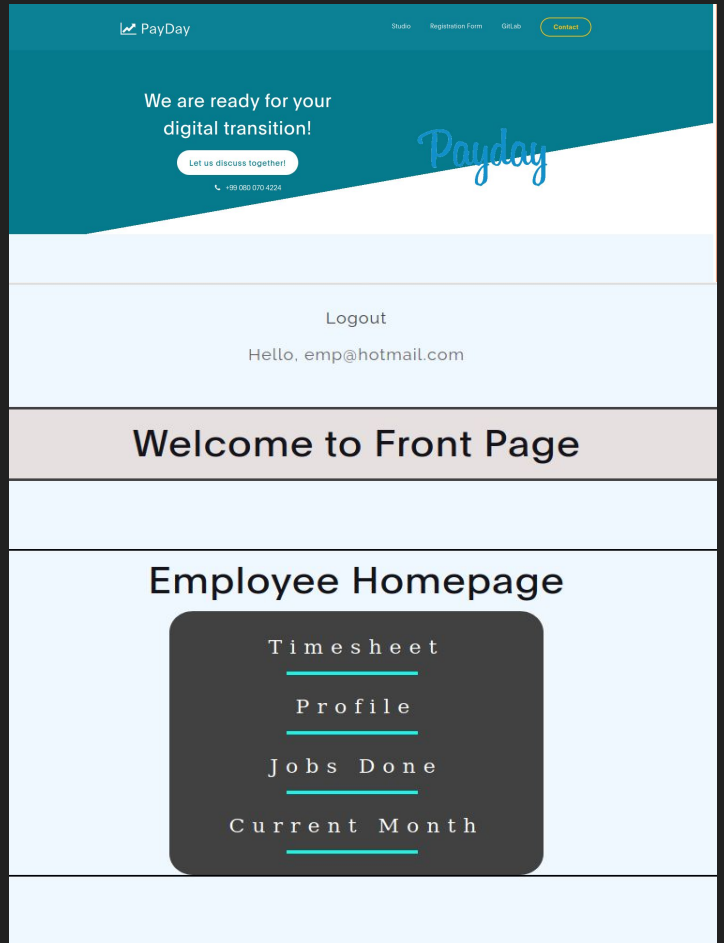
```
class ManagerRegistrationForm(UserCreationForm):
    email = forms.EmailField(max_length=60, help_text='Required. Add a valid email address',widget=forms.TextInput(

        attrs = { 'class': 'form_control'}
))

    class Meta:
        model = User
        fields = ("email", "password1", "password2", "company_id")

    def __init__(self, *args, **kwargs):
        super(ManagerRegistrationForm,self).__init__(*args, **kwargs)
        self.fields['password1'].widget.attrs.update({'class' : 'form_control'})
        self.fields['password2'].widget.attrs.update({'class' : 'form_control'})
        self.fields['company_id'].widget.attrs.update({'class' : 'form_control'})
```

```
class Manager(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, null=False)

    man_id_number   = models.CharField(primary_key=True, max_length=4, unique= True)
    first_name      = models.CharField(max_length=20)
    last_name       = models.CharField(max_length=20)
    manager         = models.BooleanField(default=True)


    objects = MyUserManager()


    def __str__(self):
        return self.man_id_number
```

User Interface Design

At first we have developed the UI “in house” by just styling it the way we want and like. However, this had issues with scalability and accessibility for mobile devices. We then decided to integrate bootstrap into our project. Not only did it fix the issues with accessibility it also made it visually more appealing.

Here is the UI for the Front Page of an Employee.



Testing

We have created test cases based on the UML use case diagram. Since we have 2 sets of users the tests are replicated for both users.

On the right is the example of a test case for Logging in as a manager.

Below is a table of tests on different devices with different OS and web browsers.

Test case ID	Login as manager		Test designed by	Vincent Lloyd Yuson		
Test priority	High		Test date	15/05/2020		
Module name	PayDay manager log in screen		Test executed by	Vincent Lloyd Yuson		
Test title	PayDay manager login		Test execution date	15/05/2020		
Description	Test PayDay manager login screen					
Pre-conditions	User has been registered as an manager					
Dependencies						
Step	Test steps	test data	Expected results	Actual results	Status	Notes
1	Navigate to manager login page		Site should open	As expected	Pass	
2	Provide email	email = testmanager@ya	credentials can be entered	As expected	Pass	
3	Provide password	Test	credentials can be entered	As expected	Pass	
4	Click login button		User is logged into their account	As expected	Pass	

UI Design Accessibility	Google chrome	Edge	Firefox	Samsung Internet
Windows	Yes	Yes	Yes	N/A
Linux	Yes	N/A	Yes	N/A
Android	Yes	N/A	N/A	Yes

Challenges

Payday

- User registration
  - 2 users - Employee and Manager
  - Need to be able to differentiate between the two users
  - Ensure Employee can’t login as a Manager
  - User accessibility
- Increasing complexity in developing the product
  - First time for either team member taking on a project of this size
- UI challenge of accessibility
  - Trying to make PayDay accessible to different types of devices for different browsers.
  - Styling forms through Forms.py

Payday

PayDay

Thank you for listening!
Any questions ?