CSE – 180006

Developing and Publishing a Date-picker Calendar Package for Websites Powered by Django

django-bootstrap-datepicker-plus

A Project Report Submitted By

Md. Abdul Monim
ID: CSE1401001008

In Partial Fulfillment of the Requirements for the degree of

Bachelor of Computer Science and Engineering



Department of Computer Science and Engineering

Faculty of Science and Engineering

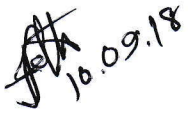
Sonargaon University (SU).

Submission Date: 10th September, 2018

Supervisor Declaration

This is to certify that project report on **Developing and Publishing a Date-picker Calendar Package for Websites Powered by Django** has been carried out by **Md. Abdul Monim**, student ID: **CSE1401001008**, student of Department of Computer Science and Engineering, Sonargaon University (SU) as a partial fulfillment of the requirement of CSE-400 Project/Thesis. The report has been prepared under my guidance and is a record of the bona fide work carried out successfully. To the best of my knowledge and as per his declaration, no parts of this report has been submitted anywhere for any degree, diploma or certificate.

Now he is permitted to submit the report. I wish him all success in his future endeavors.



Project Supervisor

Md. Monzurul Amin Ifath

Lecturer

Department of Computer Science and Engineering

Sonargaon University (SU)

Student Declaration

I Md. Abdul Monim, student of Department of Computer Science and Engineering, Sonargaon University (SU), declaring that this project paper on the stated topic has only been prepared for the fulfillment of CSE400 Project/Thesis, as the partial fulfillment of "Bachelor of Computer Science and Engineering" degree.

It has not been prepared for any other purposes, rewards, or presentation.

Md. Abdul Monim
10-09-18

MD. Abdul Monim

ID: CSE1401001008

Department of Computer Science and Engineering

Sonargaon University (SU)

Acknowledgement

In the name of ALLAH who is the most Merciful and the most Graceful.

First of all I sincerely like to thank **Md. Monzurul Amin Ifath**, Lecturer, Department of Computer Science and Engineering, Sonargaon University, giving me the opportunity to complete my report and project. I was able to make this report effectively and properly only for his right direction. Beside that I again like to thank him to give me an opportunity to submit this report.

I would like to thank my family, friends for their mental support throughout this work. I would also like to thank our Vice Chancellor **Professor M.A Razzaque** who has given us the opportunity to study in such a decent environment.

Our success could not turn into reality without these people who help us in different ways. I also like to thank all of our friends for their valuable suggestion and comments.

Objective

To develop a django-package that provides date-picker, time-picker, date-time-picker, date-range-picker, month-picker, year-picker calendar for django forms which can take away a lot of development work, compatible to any design environment a developer has to deal with. Django packages are reusable apps which can be plugged into any django project providing developers hassle-free way to use the same piece of code in multiple projects without having to reinvent the wheel or resort to dirty copy pasting pieces of code.

Table of Contents

Chapter 01: Introduction
1.1 Introduction of Project....4
1.2 Aim of Project....4
1.3 System Study & Analysis....5
1.3.1 System Analysis....5
1.4 Methodology of the project....6
1.4.1 Agile....7
1.4.2 Cause of using agile methodology....8
1.4.3 Advantages of Agile model....9
1.4.4 Listed below are a few disadvantages of Agile....9

Chapter 02: Existing System
2.1 Process of Existing System....10
2.1.1 Use existing obsolete django-packages....10
2.1.2 Use Primitive way of including a date-picker....10
2.2 Problems with Existing System....11
2.2.1 Problems with Existing django-widgets....11
2.2.2 Problems with Primitive system....11

Chapter 03: Proposed System & Design
3.1 Aim of Proposed System....12
3.1.1 Advantages of the Proposed System....13
3.1.1.1 Advantages over Primitive System....13
3.1.1.2 Advantages over Existing Packages....13
3.1.2 Drawbacks of the Proposed System....13
3.2 Why develop this django-package....14
3.2.1 Absence of the right product....14
3.2.1.1 What is a complete package....14
3.2.2 Developer's demand....15
3.2.3 Community appreciation....16
3.3 Proposed System Design....17

Chapter 04: Implementation of Model
4.1 Present django Request-Response flow with date-time fields....18
4.1.1 URL dispatcher....18
4.1.2 View....19
4.1.3 Form....19
4.1.4 Form Fields....20
4.1.5 Widgets....21
4.2 Request-Response flow of django date-time fields with django-bootstrap-datepicker-plus....22
4.3 Class diagram of django built-in widgets....23
4.3.1 DateTimeBaseInput class....23
4.3.2 DateInput class....23
4.3.3 DateTimeInput class....23
4.3.4 TimeInput class....24
4.4 Class diagram of django-bootstrap-datepicker- plus....25
4.4.1 DatePickerBaseInput class....25
4.4.2 DatePickerInput class....26

4.4.3 DateTimePickerInput class....26
4.4.2 TimePickerInput class....26
4.4.2 MonthPickerInput class....27
4.4.2 YearPickerInput class....27
4.5 Components of widget....28
4.5.1 Classes....28
4.5.2 Templates....29
4.5.3 Resources (JavaScripts/CSS)....29
4.6 Directory structure of django-bootstrap-datepicker-plus....30
4.7 System Requirements....31
4.7.1 Software Requirements....31
4.7.2 Hardware Requirements....31

Chapter 05: System Implementation
5.1 Creating the base class....32
5.1.1 The DatePickerDictionary....32
5.1.2 The DatePickerBaseInput class....33
5.1.3 Including the resources....33
5.1.4 Date format conversion between JavaScript and Python....34
5.1.5 The Constructor....35
5.1.6 Methods to link two date-picker....36
5.2 Extending the base class to DatePickerInputs....37
5.2.1 Define properties of the sub-classes....37
5.2.1 Exporting the DatePickerInput Classes....38
5.3 Publish to a Github repository....39
5.3.1 Objectives....39
5.4 Publishing to PyPI....40
5.4.1 Objectives....40
5.5 Publishing to Django Packages....41
5.5.1 Objectives....41
5.6 Publish Demo Pages....42
5.6.1 Demo with Bootstrap 3....42
5.6.2 Demo with Bootstrap 4....43
5.7 Publish package documentation (README)....44
5.7.1 The Introduction....44
5.7.2 Demo, Getting Started....45
5.7.3 Usage....46
5.7.4 Customising the Options....47
5.7.5 Licence and Acknowledgements....47

Chapter 06: Testing & Quality Management
6.1 Unit-Testing....48
6.1.1 Unit-Testing tools....48
6.1.2 Code for Testing....49
6.1.3 Unit-Testing with pytest....50
6.1.4 Unit-Testing with Travis-CI....51
6.1.4.1 Travis-CI build flow....51
6.1.4.2 Travis-CI Result....51
6.1.4.3 Travis-CI build history....52
6.2 Code-Coverage....53

6.2.1 Coveralls.io stats....53
6.2.2 Coveralls.io details and builds....54
6.3 Maintainability....55
6.3.1 Code-Climate Report....55

Chapter 07: Project Success
7.1 Project Stats on Github....56
7.1.1 Watchers....56
7.1.2 Stars....57
7.1.3 Forks....58
7.1.3.1 Clone Graph....58
7.1.4 Visitor Graph....59
7.2 Activity on Stack-Overflow....60
7.2.1 Questions on Stack-Overflow....60
7.2.2 Answers on Stack-Overflow....61

Chapter 08: Conclusion
8.1 The Experience....62
8.2 Future Work....62
8.3 Conclusion....63
8.4 References....63

Chapter 08: Tables and Figures
Figure 1.1: Stages of System Analysis 5
Figure 1.2: Agile Methodology work-flow 7
Figure 3.1: Stack-overflow questions on django date-pickers 15
Figure 3.2: Reddit post on the project 16
Figure 4.1: django Request-Response flow 18
Figure 4.2: Request-Response flow with django-bootstrap-datepicker-plus 22
Figure 4.3: Class diagram of django built-in widgets 23
Figure 4.4: Class diagram of django-bootstrap-datepicker- plus 25
Figure 4.5: Component diagram of django-bootstrap-datepicker-plus 28
Figure 4.6: Directory structure of django-bootstrap-datepicker-plus 30
Figure 5.1: The GitHub page of the Package 41
Figure 5.2: The PYPI page of the Package 40
Figure 5.3: The Django Packages page of the Package 41
Figure 5.4: The demo page with bootstrap 3 42
Figure 5.5: The demo page with bootstrap 4 43
Figure 6.1: pytest unit-test results 48
Figure 6.2: travic-CI build result 51
Figure 6.3: travic-CI build log 52
Figure 6.4: coveralls.io report 54

Chapter 01

Introduction

1.1 Introduction of Project

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so developers can focus on writing app without needing to reinvent the wheel. Many of top level sites/services are built with Django, of them notable mentions are ***Disqus, Bitbucket, Intagram and Pinterest***.

As the most popular Python Web framework, it is gaining a lot of traction. A growing number of new developers are learning this framework and plenty of projects focused on rapid scalable development are picking up django as their base framework.

In this project I am going to build a django-package that provides calendars to all types of date-time fields a developer need to integrate into a django project, fork it on Github, the biggest online repository management service and publish the package on PyPI.

1.2 Aim of Project

Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites. Django has packages named *django-bootstrap3* and *django-bootstrap4* to develop django powered sites designed with Bootstrap 3 and 4 UI frameworks respectively.

One very useful feature both the packages missed is the bootstrap date-picker widget that is often needed to develop date-time related input fields in responsive web forms designed with bootstrap. There is no out of the box feature in the packages to show a date-picker calendar, so developers have to look for third-party packages to incorporate this feature in their django powered sites.

The aim of this project is to develop that missing feature as an open-source package and host it on the PyPI (Python Packages Index) for all django developers offering a complete package needed to work with any type of date-time related fields in their django projects providing the full-featured all-in-one solution the django community is looking for over the course of last couple of years.

1.3 System Study & Analysis

1.3.1 System Analysis

System analysis is an explicit formal inquiry carried out to help someone identify a better course of action and make a better decision than he might otherwise have made. Systems analysis is a problem solving technique that decomposes a system into its component pieces for the purpose of the studying how well those component parts work and interact to accomplish their purpose. System analysis is a process of gathering and interpreting facts, diagnosing problems and the information to recorded improvements on the system. It is a problem solving activity that requires intensive communication between the system user and system developer. System analysis or study is an important phase of any system development process. A detailed study of the process must be made by various techniques like interviews, questionnaires etc. The data collected by these sources must be scrutinized to arrive to conclusion every synthesis is built upon the results of a preceding analysis and every analysis requires a subsequent. The characteristic attributes of a problem situation where systems analysis is called upon are complexity of the issue and uncertainty of the outcome of any course of action that might reasonably be taken. That part or aspect of systems analysis that concentrates on finding out wherever and indeed course of scion violates any constraints is referred to as feasibility analysis.

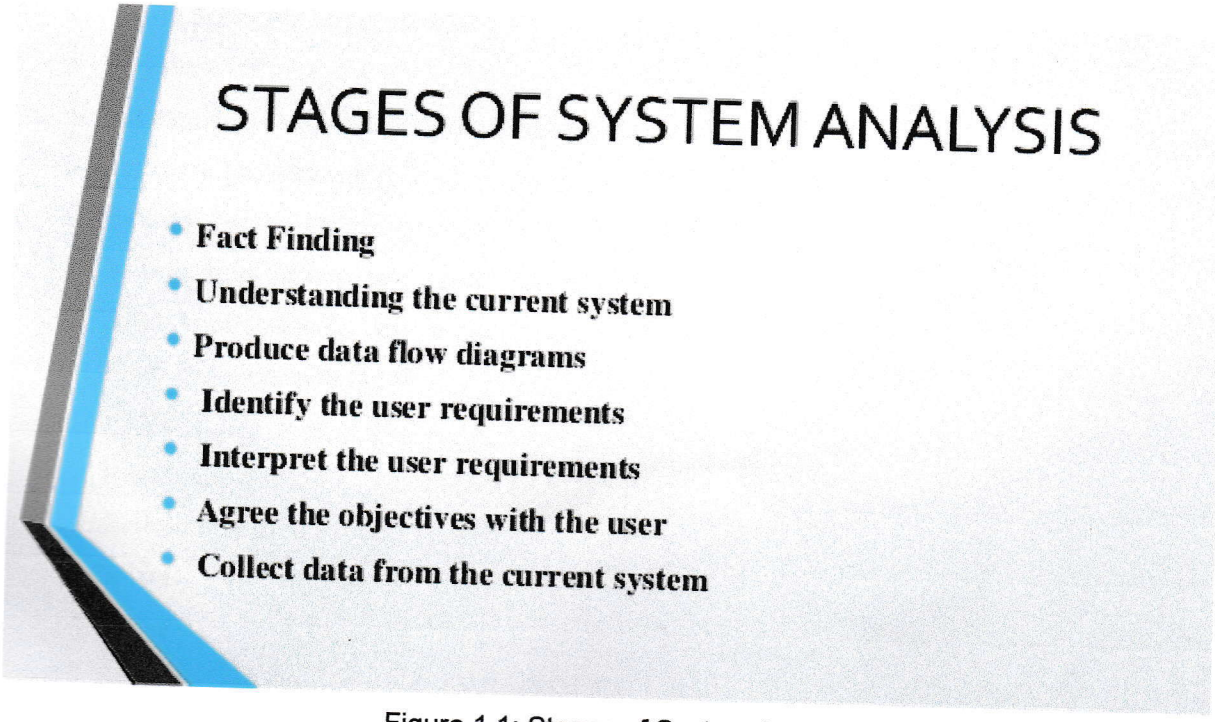


Figure 1.1: Stages of System Analysis

1.4 Methodology of the project

Methodology is the systematic, theoretical analysis of the methods applied to a field of study. It comprises the theoretical analysis of the body of methods and principles associated with a branch of knowledge. Typically, it encompasses concepts such as paradigm, theoretical model, phases and quantitative or qualitative techniques.

A methodology does not set out to provide solutions - it is, therefore, not the same as a method. Instead, a methodology offers the theoretical underpinning for understanding which method, set of methods, or best practices can be applied to specific case, for example, to calculate a specific result.

The methodology is the general research strategy that outlines the way in which research is to be undertaken and, among other things, identifies the methods to be used in it. These methods, described in the methodology, define the means or modes of data collection or, sometimes, how a specific result is to be calculated. Methodology does not define specific methods, even though much attention is given to the nature and kinds of processes to be followed in a particular procedure or to attain an objective.

The following are the common methodologies used in project management:

1. Adaptive Project Framework
2. Agile Software Development
3. Crystal Methods
4. Dynamic Systems Development Model (DSDM)
5. Extreme Programming (XP)
6. Feature Driven Development (FDD)
7. Information Technology Infrastructure Library (ITIL)
8. Joint Application Development (JAD)
9. Lean Development (LD)
10. PRINCE2
11. Rapid Application Development (RAD)
12. Rational Unified Process (RUP)
13. Scrum
14. Spiral
15. Systems Development Life Cycle (SDLC)
16. Waterfall (Traditional)

1.4.1 Agile

Agile software development refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams. Agile methods or Agile processes generally promote a disciplined project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices intended to allow for rapid delivery of high-quality software, and a business approach that aligns development with customer needs and company goals.

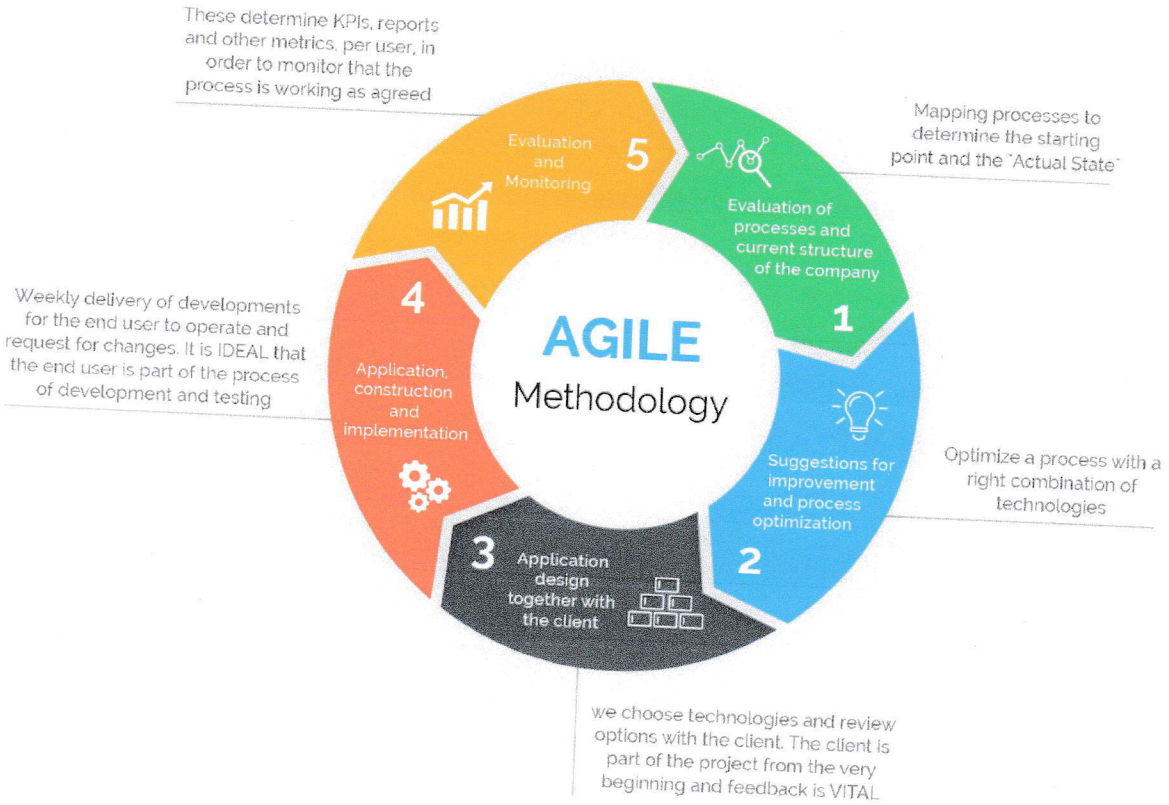


Figure 1.2: Agile Methodology work-flow

It works by breaking projects down into little bits of user functionality called user stories, prioritizing them, and then continuously delivering them in short two week cycles called iterations.

1.4.2 Cause of using agile methodology

Agile Methodology is a type of project management process. The agile method anticipates change and allows for much more flexibility than traditional methods. Clients can make small objective changes without huge amendments to the budget or schedule. The process involves breaking down each project into prioritized requirements, and delivering each individually within an iterative cycle. Iteration is the routine of developing small sections of a project at a time. Each iteration is reviewed and assessed by the development team and client. The insights gained from the assessment are used to determine the next step in development. Clients come to pre-scheduled regular meetings to review the work completed the previous iteration, and to plan work for the upcoming iteration. Detailed goals are set in each iteration meeting such as; expected changes, time estimates, priorities and budgets.

The agile method is based on giving high priority to customer participation, from the very beginning of the development cycle. The objective is to keep the client involved at every step so that they have a product that they are happy with at the end. This method saves the client money and time because the client tests and approves the product at each step of development. If there are defects or challenges, then changes can be made during production cycles to fix the issue. Traditional models of project management would not find defects as early because they do not test as often. Typically (in traditional methods of production) defects that are not discovered at the different stages can find their way into the final product. This can result in increased overhead prices and client dissatisfaction.

Businesses have proven this model of project management with their increased client satisfaction rate. The values for businesses that use this model include:

- Lower Cost
- Enables clients to be happier with the end product by making improvements and involving clients with development decisions throughout the process.
- Encourages open communication among team members, and clients.
- Providing teams with a competitive advantage by catching defects and making changes throughout the development process, instead of at the end.
- Speeds up time spent on evaluations since each evaluation is only on a small part of the whole project.
- Ensures changes can be made quicker and throughout the development process by having consistent evaluations to assess the product with the expected outcomes requested.
- It keeps each project transparent by having regular consistent meetings with the clients and systems that allow everyone involved to access the project data and progress.

Businesses use this model of project management to ensure that throughout the process

customers save time, money, and have the flexibility to make changes anytime during the development process.

1.4.3 Advantages of Agile model

- Customer satisfaction by rapid, continuous delivery of useful software.
- People and interactions are emphasized rather than process and tools. Customers, developers and testers constantly interact with each other.
- Working software is delivered frequently (weeks rather than months).
- Face-to-face conversation is the best form of communication.
- Close daily cooperation between business people and developers.
- Continuous attention to technical excellence and good design.
- Regular adaptation to changing circumstances.
- Even late changes in requirements are welcomed.

1.4.4 Listed below are a few disadvantages of Agile

- In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.
- There is lack of emphasis on necessary designing and documentation.
- The project can easily get taken off track if the customer representative is not clear what final outcome that they want.
- Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for newbie programmers, unless combined with experienced resources.

Agile Developments also fails at times due to Unrealistic Expectations – Agile actually is and what it can help you achieve. Agile is commonly believed to be a set a practices, processes and tools, when in fact, Agile is really more of a mind-set and culture.

Chapter 02

Existing System

2.1 Process of Existing System

There is no developers friendly existing system. Existing widgets available on open-source market are outdated and not compatible with django version 2.0 and above. The only choice developers are left with is to include a date-picker widget the primitive way. Many developers choose to not go through all these hassles and provide the site users with default HTML date-input which is not even supported by all major browsers and creates a inconsistent user experience.

2.1.1 Use existing obsolete django-packages

1. Existing django packages does not support wide range of django versions so developers often look for django package that is specific to the django version they are working with.
2. If such a package is not found developers tend to write patch codes to make it work on the django version they are working on.

2.1.2 Use Primitive way of including a date-picker

The primitive way of adding a date-picker widget to a django site is detailed below:

1. A developer has to look for a date-picker widget that does not break the site layout he is working on.
2. If the site is built with Bootstrap, then he has to look for date-picker widget that goes well with the specific Bootstrap version the site is built with. Date-picker widget built for Bootstrap version 3 does not go with Bootstrap version 4.
3. The developer has to download the HTML/JS/CSS resource files associated with the date-picker widget to his project's app directory and link them up in his master layout file.
4. The developer has to write JavaScript code to initialize the date-picker widget when the page is loaded.

2.2 Problems with Existing System

2.2.1 Problems with Existing django-widgets

1. Most of the django-widgets available on open-source market is outdated, very few of them supports django version 2.0 and above.
2. The django-widgets available are not well documented or developer friendly.
3. They have pretty limited available features and customization.
4. The widgets that supports django version 2.0 does not have backward compatibility, they cannot be used on django project built on an older django version.
5. Very few of them are compatible with Bootstrap version 4, so cannot be used in a project built using bootstrap4 django-widget.
6. None of the django-widgets available supports built-in date-range picker functionality that developers often need when they need to select starting and ending date or time of an event.
7. None of the django-widgets available provide features to pick a month or a year only, that is often needed by developers
8. None of the available django-widgets is a complete toolbox for a django developer.

2.2.2 Problems with Primitive system

As a developer has to do a lot of manual work to add a date-picker plug-in into the django project he is working on he has to face a lot of troubles:

1. **Time consuming:** A developer has to spend a considerable amount of time into incorporating a date-picker widget to his project which he could use on other development features.
2. **Complicated work:** It's not easy for a new django developer to do all the configuration stuff as he can easily caught up by simple mistakes that can hold him back for hours. Even experienced developers often fall into this.
3. **Unorganised design:** Django apps are designed to be pluggable to any project, so everything is structured and organized into separate projects. Downloading and including date-picker resources (JS/CSS/images) to the project breaks that structure.

Chapter 03

Proposed System & Design

3.1 Aim of Proposed System

The aim of this project is to build a django-package that supports the latest Django version with backward compatibility upto Django version 1.8. This includes:

1. Developing a django-package that supports a wide range of django versions making it fit for both newer and older django-powered sites.
2. Developing the package so that it supports both Bootstrap version 3 and 4. Bootstrap 3 has been around for a while and many existing projects are built on it making it important to support while newer projects tend to go for the latest Bootstrap version.
3. Providing both date-picker and date-time-picker widgets including some some extra features that wraps up most of the requirements a developer might ever encounter keeping the package scalable for developing.
4. Providing date-range-picker functionality out of the box and thus eliminating the need for any additional package or writing patch codes to the package resources.
5. Publishing the repository to Github for keeping a clean track of development and receive community support in the form of suggestions, bug reports and feature requests.
6. Publishing the package to PyPI making it easily available via pip so that developers can install it right away by a simple console command.
7. Enlist the package to Django Packages index to make it more available to developers specific to Django development.
8. Writing test files encouraging writing testable code that helps better development and provides an instant build status.
9. Writing PEP8 compliant code following standard code writing practices throughout the whole project.

3.1.1 Advantages of the Proposed System

3.1.1.1 Advantages over Primitive System

1. It is pretty easy to setup, a simple installation via pip and couple configuration, one can easily get started with it in minutes.
2. The package manages all the JavaScript and CSS files itself, so the developer does not need to worry about looking for them, downloading and linking them to templates.
3. This package provides a range-picker functionality than can be setup with almost no code thus relieves the developer of so much repetitive works.
4. The package helps writing reusable code which is well-inspired by the Django community.
5. The package provide lot more customisation without even writing a line of JavaScript code.

3.1.1.2 Advantages over Existing Packages

1. None of the existing packages supports latest Django versions, most of them limited to a specific older django version. This package supports a wide range of Django versions (1.8~2.0) including the latest version released.
2. Most of the packages are specific to either Bootstrap version 3 or 4. This package supports both Bootstrap 3 and 4 thus fit for any django project even projects having both UI build system.
3. Most packages available only provides date-picker or datetime-picker. This package provides date-picker, time-picker, datetime-picker even month-picker and year-picker.
4. No packages available on market provides range-picker facility. For range-picker you need to install another package. This package includes a range-picker solution linking any two date-inputs with minimal configuration.

3.1.2 Drawbacks of the Proposed System

1. The proposed system does not support very old django versions. So it cannot be used with such older projects unless the django version is upgraded to at least version 1.8.
2. If any future Django release has some breaking changes then we need to release a new version of this package too fixing the compatibility issues.

3.2 Why develop this django-package

3.2.1 Absence of the right product

There are tons of date-time related packages on Github. Most of them are obsolete and of the few of the still working packages, none of them provides a complete package a developer needs. So let's define what is a complete package in this regard.

3.2.1.1 What is a complete package

A complete bootstrap date-picker calendar package should meet all the features and customization a developer needs. It should address if not all most of the following concerns:

1. The package should work on the latest django release providing all new projects the opportunity to use the benefit of it.
2. The package should support a variety of old django releases as most the current django projects are built on old releases.
3. The package should work with django-bootstrap 3 as it is still loved by a huge crowd of developers.
4. The package should support django-bootstrap 4 as a developer starting a new project will choose to use the latest bootstrap features.
5. It should provide date, time, date-time picker calendars as these are often used by the developers in django forms.
6. It should provide a date-range picker option so that a developer does not need to tweak the package code to provide such functionality.
7. The package should be hosted on PyPI so it can be easily installed via a simple pip command.
8. The package should provide reasonable flexibility and customizable options to modify it according to the need to the project.
9. The package should be well documented presenting instruction on how to install it, integrate it, configure it and customize it properly.

3.2.2 Developer's demand

There are more than 100 questions on stack-overflow looking for a feasible way of including a date-picker to django-powered site and none of them are well-answered. There exists zero answer on this topic that provides a solution that works on newest django versions without troubles. The django packages suggested in these questions are obsolete and needs tweaking to make it work. So the django developer community is still looking for a solution to this.

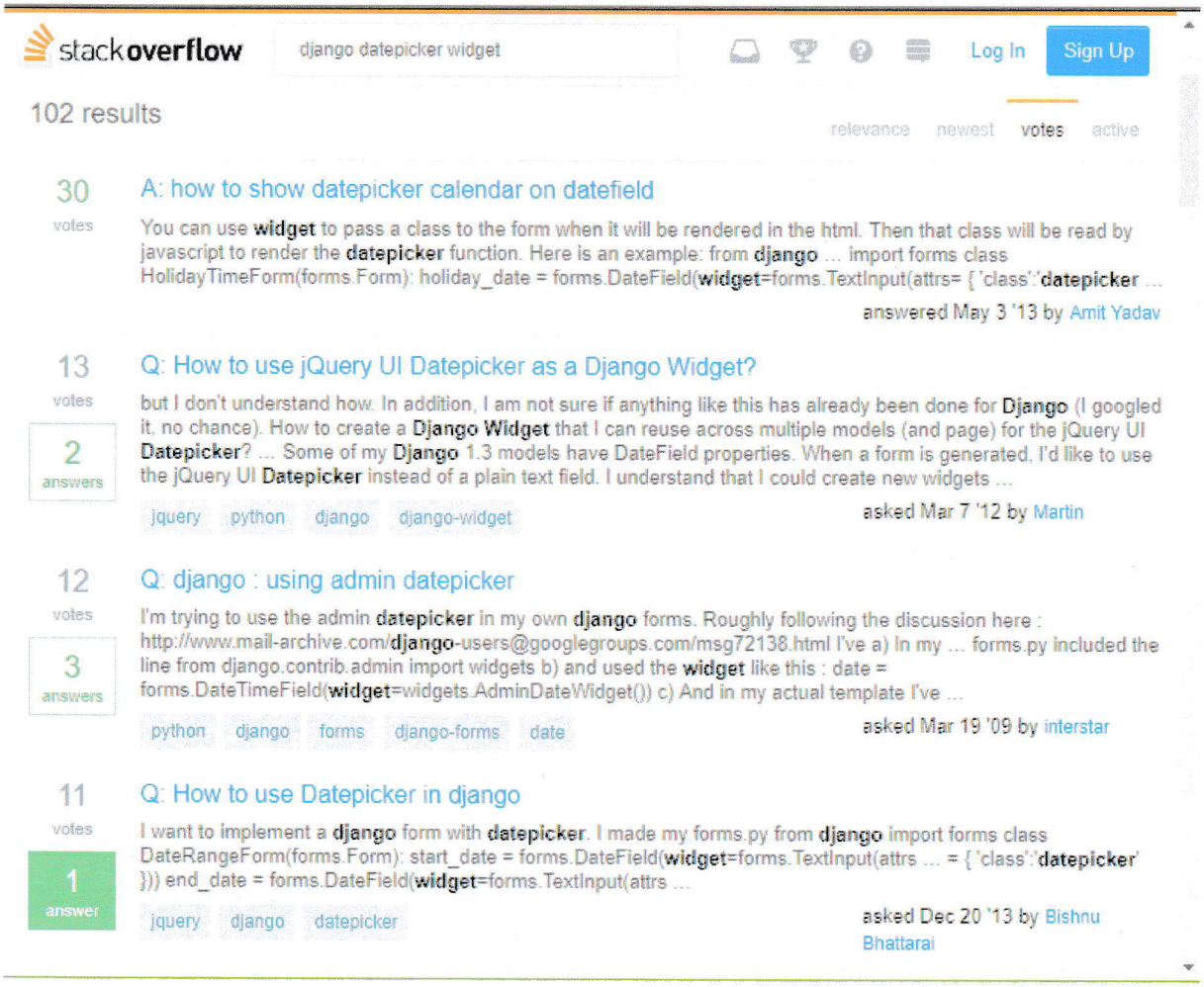


Figure 3.1: Stack-overflow questions on django date-pickers

3.2.3 Community appreciation

During the development of the package we posted it on reddit asking for suggestions. The r/django community appreciates the development of the project that can help thousands of django developers world-wide.

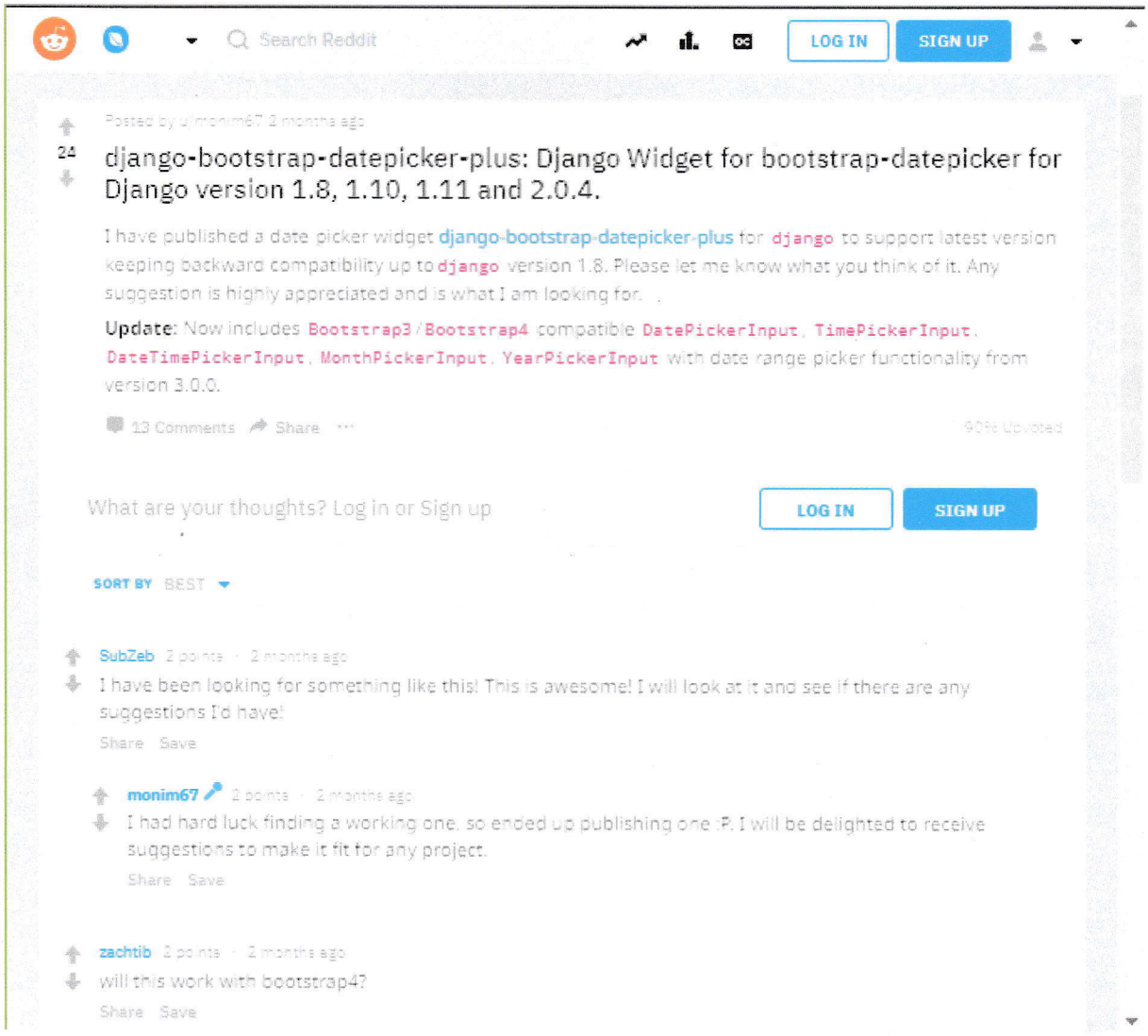


Figure 3.2: Reddit post on the project

3.3 Proposed System Design

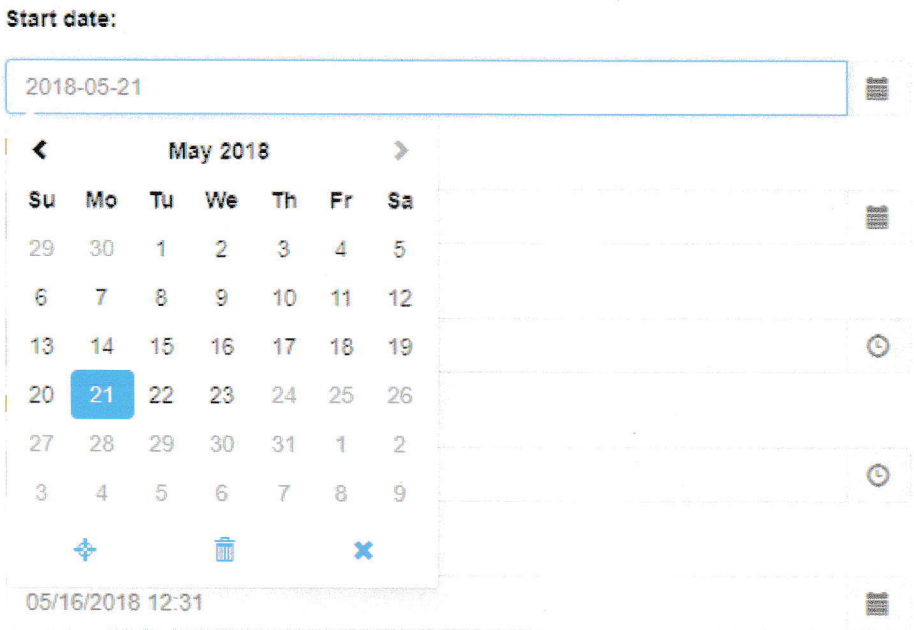
1. The package will be uploaded to PyPI to make it publicly accessible which can be installed via pip with a simple command.

```
pip install django-bootstrap-datepicker-plus
```

2. Add the package to the list of INSTALLED_APPS.

```
INSTALLED_APPS = [
    # Add the following
    'bootstrap_datepicker_plus',
]
```

3. The required DatePickerInput then can be imported to any form present in the project and bound to any date-time field as widget parameter (Details on README).
4. This will show a Bootstrap date-picker on the selected field



5. :

3.3 Proposed System Design

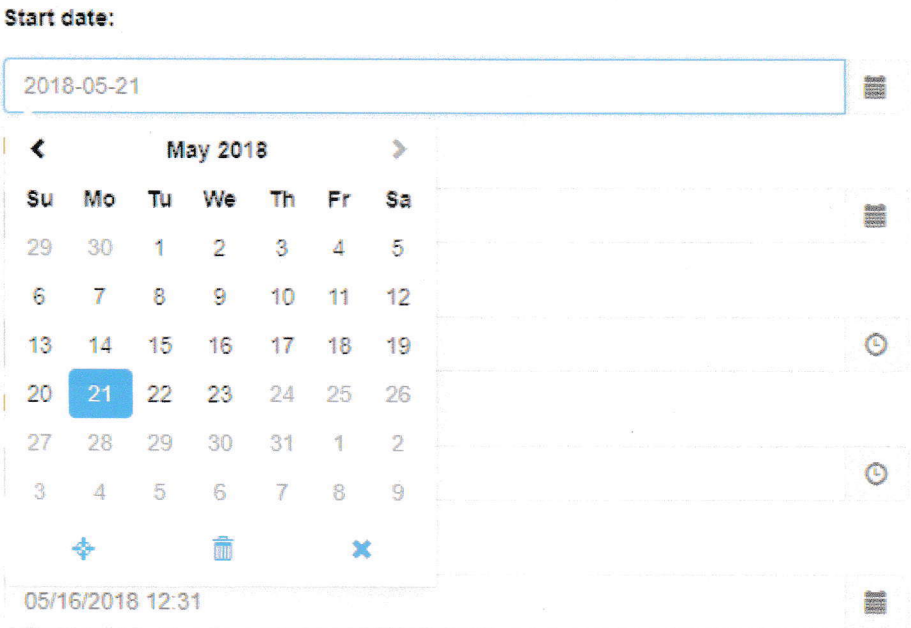
1. The package will be uploaded to PyPI to make it publicly accessible which can be installed via pip with a simple command.

```
pip install django-bootstrap-datepicker-plus
```

2. Add the package to the list of INSTALLED_APPS.

```
INSTALLED_APPS = [
    # Add the following
    'bootstrap_datepicker_plus',
]
```

3. The required DatePickerInput then can be imported to any form present in the project and bound to any date-time field as widget parameter (Details on README).
4. This will show a Bootstrap date-picker on the selected field



5. :

Chapter 04

Implementation of Model

4.1 Present django Request-Response flow with date-time fields

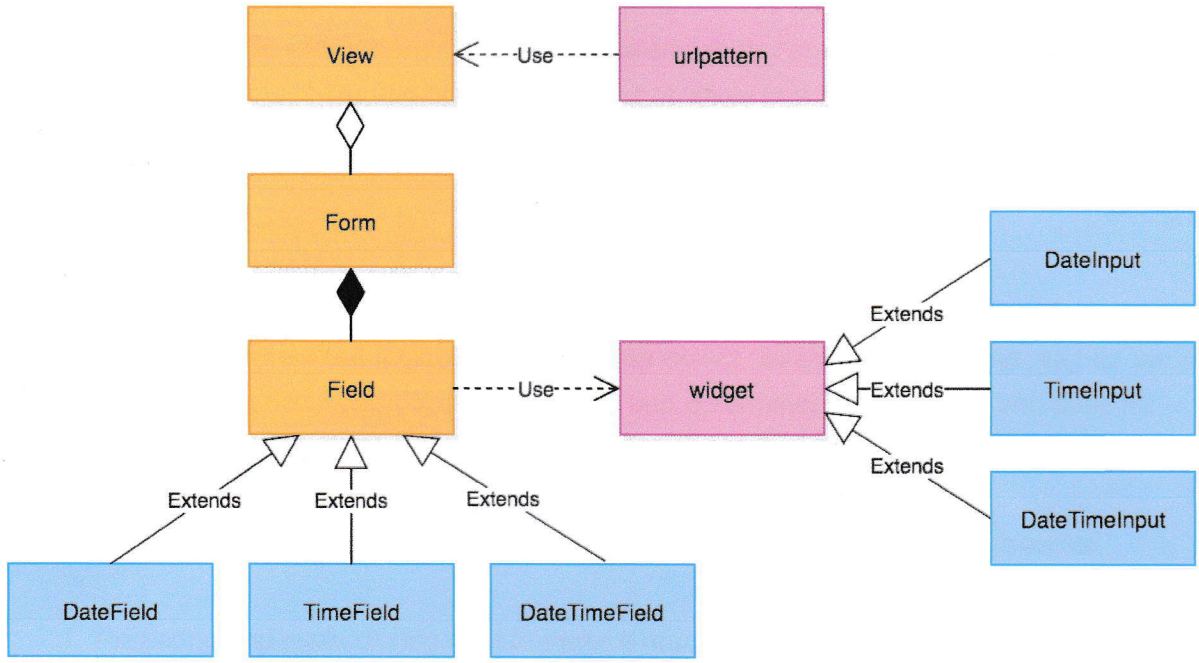


Figure 4.1: django Request-Response flow

4.1.1 URL dispatcher

When a user requests a page from a Django-powered site, the **URL dispatcher** determines which Python code to execute. If the URL patterns matches with the request, Django imports and calls the given **view**, which is a simple Python function (or a class-based view). The view gets passed the following arguments:

- An instance of *HttpRequest*.
- If the matched URL pattern returned no named groups, then the matches from the regular expression are provided as positional arguments.
- The keyword arguments are made up of any named parts matched by the path expression, overridden by any arguments specified in the optional kwargs argument to *django.urls.path()* or *django.urls.re_path()*.

4.1.2 View

A view function, or view for short, is simply a Python function that takes a Web request and returns a Web response. This response can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image . . . or anything, really. The view itself contains whatever arbitrary logic is necessary to return that response. This code can live anywhere you want, as long as it's on your Python path. There's no other requirement–no "magic," so to speak. For the sake of putting the code somewhere, the convention is to put views in a file called views.py, placed in your project or application directory.

4.1.3 Form

At the heart of this system of components is Django's Form class. In much the same way that a Django model describes the logical structure of an object, its behavior, and the way its parts are represented to us, a Form class describes a form and determines how it works and appears.

All form classes are created as subclasses of either *django.forms.Form* or *django.forms.ModelForm*. Django handles three distinct parts of work involved in forms:

1. Preparing and restructuring data to make it ready for rendering
2. Creating HTML forms for the data
3. Receiving and processing submitted forms and data from the client

4.1.4 Form Fields

Django form fields define two types of functionality, a form field's HTML markup and its server-side validation facilities. For example, a Django form field translates into the actual HTML form markup (e.g. an <input>, <select> or <textarea> tag), the HTML markup attributes (e.g. length, whether a field can be left blank or if a field must be disabled), as well as the necessary hooks to easily perform server-side validation on a form field's data.

Naturally, the forms library comes with a set of Field classes that represent common validation needs.

- BooleanField
- CharField
- ChoiceField
- TypedChoiceField
- **DateField**
- **DateTimeField**
- DecimalField
- DurationField
- EmailField
- FileField
- FilePathField
- FloatField
- ImageField
- IntegerField
- GenericIPAddressField
- MultipleChoiceField
- TypedMultipleChoiceField
- NullBooleanField
- RegexField
- SlugField
- **TimeField**
- URLField
- UUIDField

Note: The bold names are the fields that can use the widgets provided by proposed django package.

4.1.5 Widgets

A widget is Django's representation of an HTML input element. The widget handles the rendering of the HTML, and the extraction of data from a GET/POST dictionary that corresponds to the widget.

The HTML generated by the built-in widgets uses HTML5 syntax, targeting <!DOCTYPE html>. For example, it uses boolean attributes such as checked rather than the XHTML style of checked='checked'.

These are the built-in widgets available in django.forms.widgets module:

- Widgets handling input of text
    - TextInput
    - NumberInput
    - EmailInput
    - URLInput
    - PasswordInput
    - HiddenInput
    - **DateInput**
    - **DateTimeInput**
    - **TimeInput**
    - Textarea
- Selector and checkbox widgets
    - CheckboxInput
    - Select
    - NullBooleanSelect
    - SelectMultiple
    - RadioSelect
    - CheckboxSelectMultiple
- File upload widgets
    - FileInput
    - ClearableFileInput
- Composite widgets
    - MultipleHiddenInput
    - SplitDateTimeWidget
    - SplitHiddenDateTimeWidget

- SelectDateWidget

Note: The bold names are the widgets that will be replaced the widgets provided by proposed django package.

4.2 Request-Response flow of django date-time fields with django-bootstrap-datepicker-plus

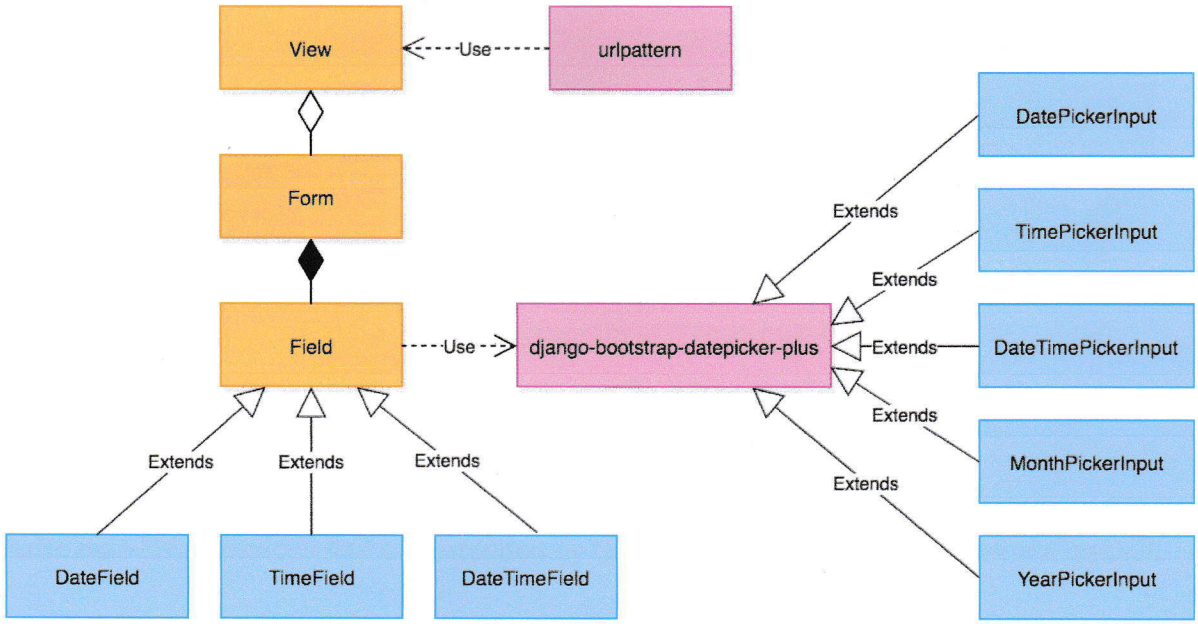


Figure 4.2: Request-Response flow with django-bootstrap-datepicker-plus

4.3 Class diagram of django built-in widgets

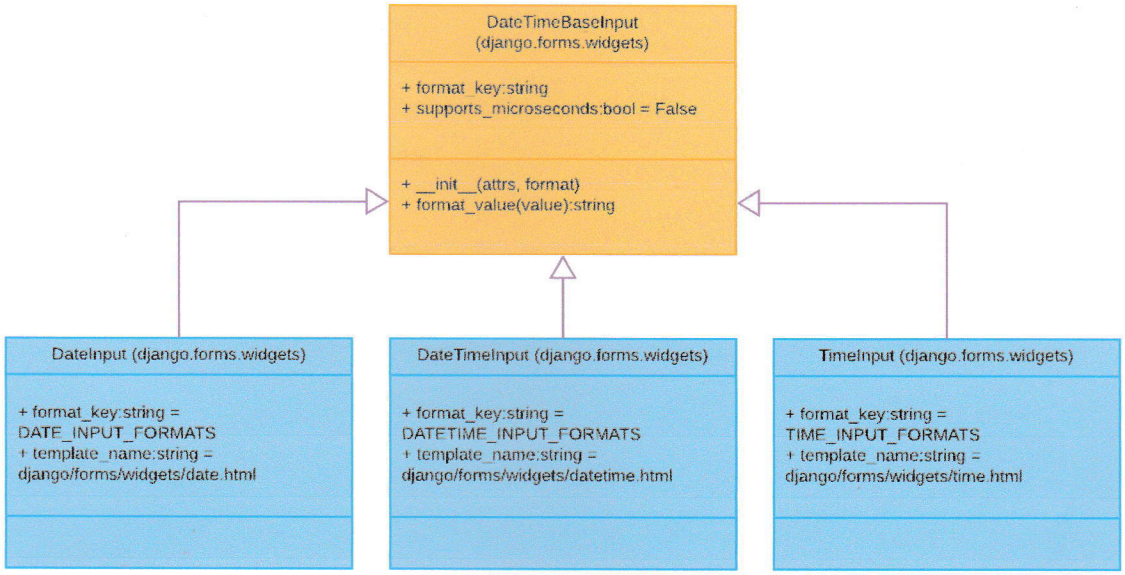


Figure 4.3: Class diagram of django built-in widgets

4.3.1 DateTimeBaseInput class

This is the base class of all date-time classes. It is a sub-class of TextInput class and holds the basic date-time functions that are extended by DateInput, DateTimeInput, TimeInput classes.

4.3.2 DateInput class

- input_type: 'text'
- template_name: 'django/forms/widgets/date.html'
- Renders as: <input type="text" ...>

This class is used for DateField type fields of django forms. It shows and validates the date-fields in django forms with the formats listed under DATE_INPUT_FORMATS key.

4.3.3 DateTimeInput class

- input_type: 'text'
- template_name: 'django/forms/widgets/datetime.html'
- Renders as: <input type="text" ...>

This class is used for DateTimeField type fields of django forms. It shows and validates the date-time fields in django forms with the formats listed under DATETIME_INPUT_FORMATS key.

4.3.4 TimeInput class

- input_type: 'text'
- template_name: 'django/forms/widgets/time.html'
- Renders as: <input type="text" ...>

This class is used for TimeField type fields of django forms. It shows and validates the date-fields in django forms with the formats listed under TIME_INPUT_FORMATS key.

4.4 Class diagram of django-bootstrap-datepicker-plus

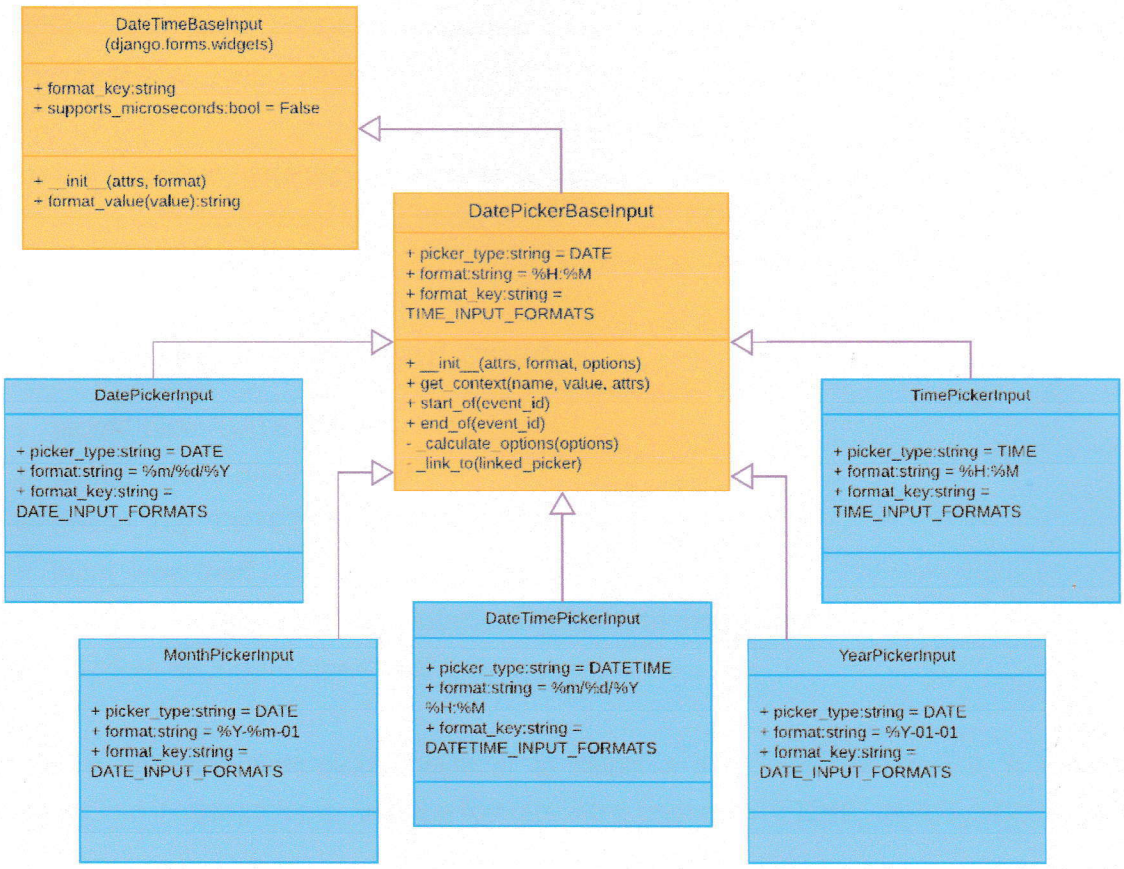


Figure 4.4: Class diagram of django-bootstrap-datepicker-plus

4.4.1 DatePickerBaseInput class

- input_type: 'text'
- template_name: 'bootstrap_datepicker_plus/time-picker.html'
- Renders as: <input type="text" ...>

4.4.2 DatePickerInput class

- picker_type: 'DATE'
- format: '%m/%d/%Y'
- format_key: 'DATE_INPUT_FORMATS'
- input_type: 'text'
- template_name: 'bootstrap_datepicker_plus/date-picker.html'
- Renders as: <input type="text" ...>

It shows a bootstrap date-picker calendar to select a date from it. It validates the DateField in django forms with the formats listed under DATE_INPUT_FORMATS key.

4.4.3 DateTimePickerInput class

- picker_type: 'DATETIME'
- format: '%m/%d/%Y %H:%M'
- format_key: 'DATETIME_INPUT_FORMATS'
- input_type: 'text'
- template_name: 'bootstrap_datepicker_plus/date-picker.html'
- Renders as: <input type="text" ...>

It shows a bootstrap date-picker calendar to select a date with time from it. It validates the DateTimeField in django forms with the formats listed under DATETIME_INPUT_FORMATS key.

4.4.2 TimePickerInput class

- picker_type: 'TIME'
- format: '%H:%M'
- format_key: 'TIME_INPUT_FORMATS'
- input_type: 'text'
- template_name: 'bootstrap_datepicker_plus/time-picker.html'
- Renders as: <input type="text" ...>

It shows a bootstrap date-picker calendar to select a time from it. It validates the TimeField in django forms with the formats listed under DATE_INPUT_FORMATS key.

4.4.2 MonthPickerInput class

- picker_type: 'DATE'
- format: '%m/%d/%Y'
- format_key: 'DATE_INPUT_FORMATS'
- input_type: 'text'
- template_name: 'bootstrap_datepicker_plus/date-picker.html'
- Renders as: <input type="text" ...>

It shows a bootstrap date-picker calendar to select a month from it. It validates the DateField in django forms with the formats listed under DATE_INPUT_FORMATS key. The month is stored as a DateField, so it can be used to search between date-ranges without any conversions.

4.4.2 YearPickerInput class

- picker_type: 'DATE'
- format: '%m/%d/%Y'
- format_key: 'DATE_INPUT_FORMATS'
- input_type: 'text'
- template_name: 'bootstrap_datepicker_plus/date-picker.html'
- Renders as: <input type="text" ...>

It shows a bootstrap date-picker calendar to select a year from it. It validates the date-fields in django forms with the formats listed under DATE_INPUT_FORMATS key.

4.5 Components of widget

The widget is composed of python classes, HTML templates and JavaScript/CSS resource files.

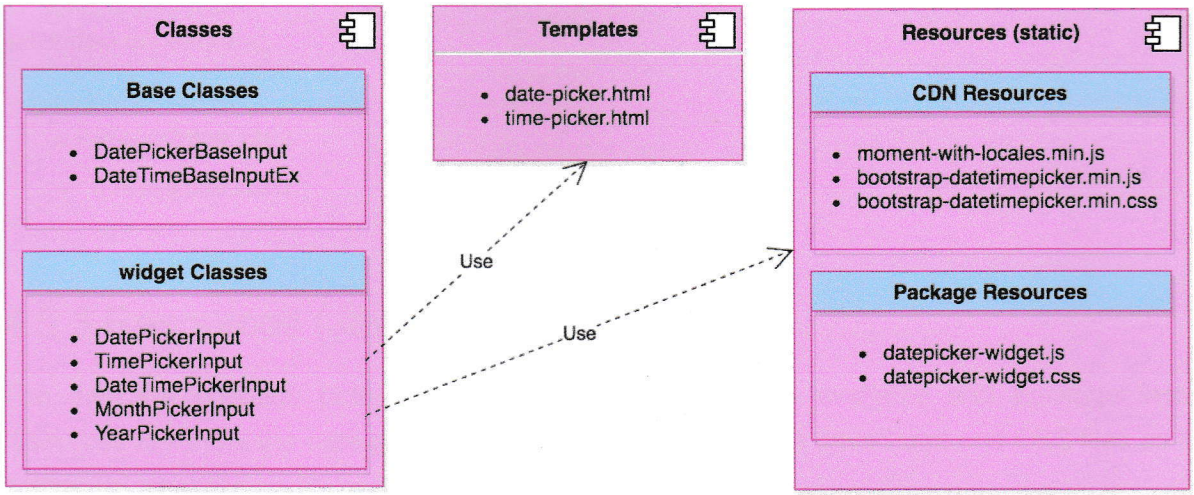


Figure 4.5: Component diagram of django-bootstrap-datepicker-plus

4.5.1 Classes

The python classes are the widget base classes and their sub-classes in respective python files with py extension.

The base classes are:

- DatePickerBaseInput
- DateTimeBaseInput

The sub-classes are:

- DatePickerInput
- DateTimePickerInput
- TimePickerInput
- MonthPickerInput
- YearPickerInput

4.5.2 Templates

Templates generates the HTML markup needed to render the date-pickers on screen. They are available in widgets since django version 1.11. But this package includes fallbacks for django versions older than 1.11. The following templates are included in the package which can be extended on demand.

- date-picker.html
- time-picker.html

4.5.3 Resources (JavaScripts/CSS)

The package contains both CDN hosted resources where available and package included resources to render the date-picker. The resources included are JavaScript and CSS files as listed below.

The CDN-hosted resources are:

- moment-with-local.min.js
- bootstrap-datetimepicker.min.js
- bootstrap-datetimepicker.min.css

The package-included resources are:

- datepicker-widget.js
- datepicker-widget.css

4.6 Directory structure of django-bootstrap-datepicker-plus

The package packed in bootstrap_datepicker_plus directory. The python classes are in the root of the directory. The templates are included in a templates directory as per the conversion of django app development. All the package included resources are located in the static folder.

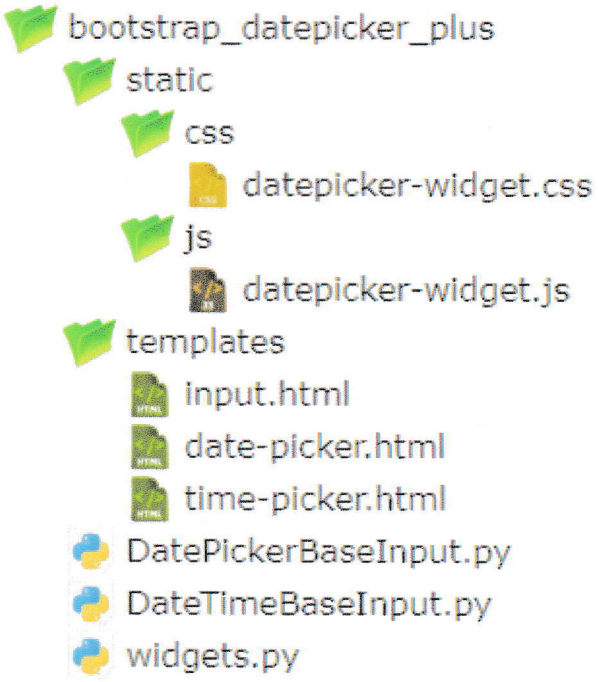


Figure 4.6: Directory structure of django-bootstrap-datepicker-plus

4.7 System Requirements

4.7.1 Software Requirements

The package is written in Python 3, and supports any latest version of django including older django versions back to 1.8. It works with both Bootstrap 3 and 4. It needs jQuery to render the date-picker. The complete list of requirements are as follows.

- Python >= 3.3
- Django >= 1.8
- Bootstrap >= 3
- jquery >= 1.7.1

4.7.2 Hardware Requirements

The package can be used with any server capable of running a django-powered site running smoothly. There is no special hardware requirements for this package.

Chapter 05

System Implementation

5.1 Creating the base class

5.1.1 The DatePickerDictionary

1. The DatePickerDictionary class keeps the record of all the DatePickerInput classes used by the whole site.
2. The generate_id method to generate unique ID for each DataPicketInput class.
3. The get_base_input method to check of Django version used and return DateTimeBaseInput if older django version is used.

DatePickerBaseInput.py

```
# -*- coding: utf-8 -*-
from json import dumps as json_dumps


class DatePickerDictionary:
    _i = 0
    items = dict()

    @classmethod
    def generate_id(cls):
        cls._i += 1
        return 'dp_%s' % cls._i

    @staticmethod
    def get_base_input(test=False):
        from django.forms.widgets import DateTimeBaseInput
        if 'get_context' in dir(DateTimeBaseInput) and not test:
            # django version 1.11 and above
            return DateTimeBaseInput
        else:
            # django version below 1.11
            from .DateTimeBaseInputEx import DateTimeBaseInputEx
            return DateTimeBaseInputEx

```

5.1.2 The DatePickerBaseInput class

1. It extends the BaseInput provided by DatePickerDictionary.
2. It has the default template_name, picker_type and format.
3. The config property holds the configurations of the DatePicker
4. The options property holds the options which are passed directly to JavaScript date-picker instance.

```
class DatePickerBaseInput(DatePickerDictionary.get_base_input()):
    template_name = 'bootstrap_datepicker_plus/date-picker.html'
    picker_type = 'DATE'
    format = '%m/%d/%Y'
    config = {}
    _default_config = {
        'id': None,
        'picker_type': None,
        'linked_to': None,
        'options': {}  # final merged options
    }
    options = {}  # options extended by user
    options_parameter = {}  # options passed as parameter
    _default_options = {
        'showClose': True,
        'showClear': True,
        'showTodayButton': True,
    }
```

5.1.3 Including the resources

1. The Media class of the DatePickerInput holds the list of all JavaScript and CSS resources that are needed to include in the HTML output.

```
class Media:
    js = (
        'https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.9.0/moment-with-locales.min.js',
        'https://cdnjs.cloudflare.com/ajax/libs/bootstrap-datetimepicker/4.17.47/js/bootstrap-datetimepicker.min.js',
        'bootstrap_datepicker_plus/js/datepicker-widget.js'
    )
    css = {'all': (
        'https://cdnjs.cloudflare.com/ajax/libs/bootstrap-datetimepicker/4.17.47/css/bootstrap-datetimepicker.css',
        'bootstrap_datepicker_plus/css/datepicker-widget.css'
    ), }
```

5.1.4 Date format conversion between JavaScript and Python

1. The format_map property holds a mapping of moment date-time format identifiers to python date-time format identifiers.
2. The format_py2js method converts a python date-time format to a moment date-time format.
3. The format_js2py method converts a moment date-time format to a python date-time format.

```
        format_map = (
            ('DDD', r'%j'),
            ('DD', r'%d'),
            ('MMMM', r'%B'),
            ('MMM', r'%b'),
            ('MM', r'%m'),
            ('YYYY', r'%Y'),
            ('YY', r'%y'),
            ('HH', r'%H'),
            ('hh', r'%I'),
            ('mm', r'%M'),
            ('ss', r'%S'),
            ('a', r'%p'),
            ('ZZ', r'%z'),
        )

    ⊞   class Media:…

        @classmethod
        def format_py2js(cls, format):
            ''' convert python datetime format to moment format '''
            for js, py in cls.format_map:
                format = format.replace(py, js)
            return format

        @classmethod
        def format_js2py(cls, format):
            ''' convert moment format to python datetime format '''
            for js, py in cls.format_map:
                format = format.replace(js, py)
            return format
```

5.1.5 The Constructor

1. The __init__ method is the constructor method of the DatePickerBaseInput. Accepts the following arguments
    1. **self:** the initialized object instance.
    2. **attrs:** the HTML attributes of the input tag that will render the date-picker.
    3. **format:** the python date-time format
    4. **options:** the options to customize the JavaScript date-picker instance
2. The _calculate_options is a internal method that calculate the options of the date-picker provided by the user overriding the default options
3. The get_context method is called by Django before rendering the date-picker. The context returned contains the HTML attributes of the input element.

```
    def __init__(self, attrs={}, format=None, options={}):
        self.config = self._default_config.copy()
        self.config['id'] = DatePickerDictionary.generate_id()
        self.config['picker_type'] = self.picker_type
        self.config['options'] = self._calculate_options(options)
        # Configure Format
        if format is None:
            format = self.format
        if self.config['options'].get('format'):
            format = self.format_js2py(self.config['options'].get('format'))
        else:
            self.config['options']['format'] = self.format_py2js(format)
        # Initilize
        if 'class' not in attrs:
            attrs['class'] = 'form-control'
        super(DatePickerBaseInput, self).__init__(attrs, format)

    def _calculate_options(self, options):...

    def get_context(self, name, value, attrs):
        context = super(DatePickerBaseInput, self).get_context(
            name, value, attrs)
        context['widget']['attrs']['dp_config'] = json_dumps(self.config)
        return context
```

5.1.6 Methods to link two date-picker

1. The start_of method takes in the event_id of an event to create link-up between two DatePickerInputs and sets the DatePickerInput as the start date of the event.
2. The end_of method takes in the event_id of an event to create link-up between two DatePickerInputs and sets the DatePickerInput as the end date of the event.

```
    def start_of(self, event_id):
        '''
        Set Datepicker as the start-date of a date-range

        event_id => a user-defined unique string to identify the date-range
        '''
        DatePickerDictionary.items[str(event_id)] = self
        return self

    def end_of(self, event_id):
        '''
        Set Datepicker as the end-date of a date-range

        event_id => a user-defined unique string to identify the date-range
        '''
        event_id = str(event_id)
        if event_id in DatePickerDictionary.items:
            linked_picker = DatePickerDictionary.items[event_id]
            self.config['linked_to'] = linked_picker.config['id']
            self.config['options'].update(linked_picker.config['options'])
            if self.options_parameter:
                self.config['options'].update(self.options_parameter)
            # Important! See https://github.com/Eonasdan/bootstrap-datetimepicker/issues
            self.config['options']['useCurrent'] = False
            self._link_to(linked_picker)
        else:
            raise KeyError(
                'start-date not specified for event_id "%s"' % event_id)
        return self
```

5.2 Extending the base class to DatePickerInputs

5.2.1 Define properties of the sub-classes

1. Create widgets.py file to contain all the sub-classes.
2. All classes will extend the BaseDatePickerInput class.
3. Classes will override the default values to render different date-picker widgets.

widgets.py

```
from .DatePickerBaseInput import DatePickerBaseInput


class DatePickerInput(DatePickerBaseInput):
    picker_type = 'DATE'
    format = '%m/%d/%Y'
    format_key = 'DATE_INPUT_FORMATS'


class TimePickerInput(DatePickerBaseInput):
    picker_type = 'TIME'
    format = '%H:%M'
    format_key = 'TIME_INPUT_FORMATS'
    template_name = 'bootstrap_datepicker_plus/time-picker.html'

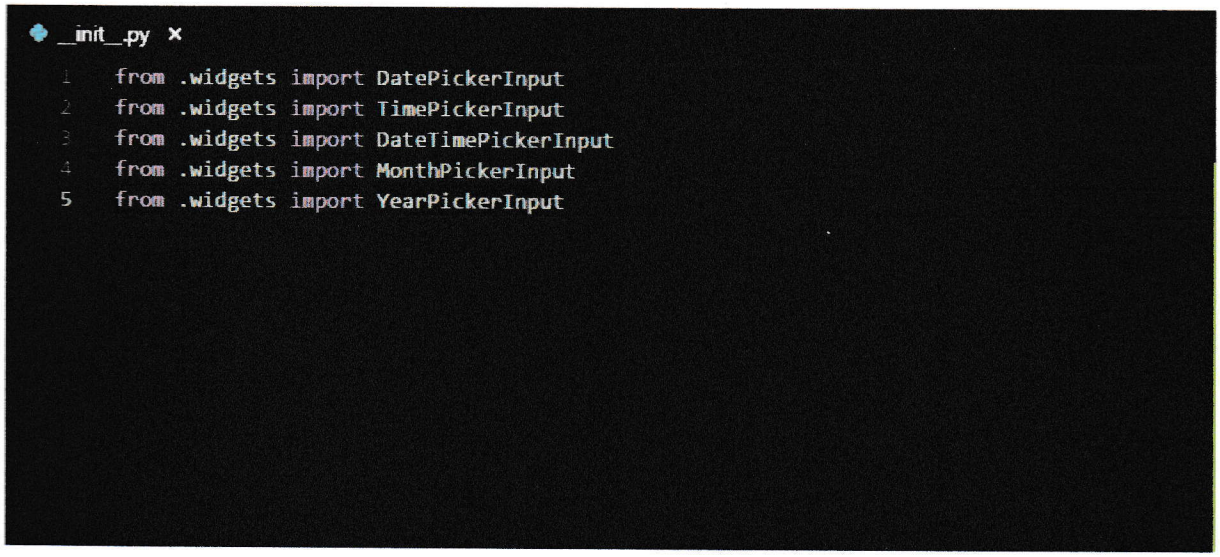

class DateTimePickerInput(DatePickerBaseInput):
    picker_type = 'DATETIME'
    format = '%m/%d/%Y %H:%M'
    format_key = 'DATETIME_INPUT_FORMATS'


class MonthPickerInput(DatePickerBaseInput):
    picker_type = 'MONTH'
    format = '%Y-%m-01'
    format_key = 'DATE_INPUT_FORMATS'


class YearPickerInput(DatePickerBaseInput):
    picker_type = 'YEAR'
    format = '%Y-01-01'
    format_key = 'DATE_INPUT_FORMATS'
```

5.2.1 Exporting the DatePickerInput Classes

Importing into __init__.py makes selected Classes, functions, etc accessible to the package level so they can be conveniently imported from the package. So we will import the DatePickerInput classes to __init__.py file to make it accessible outside of the package.



5.3 Publish to a Github repository

GitHub Inc. is a web-based hosting service for version control using Git. It is mostly used for computer code. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features.

5.3.1 Objectives

1. Create a fresh repository [1]
2. Short, memorable but self-explanatory name for the repository
3. Add a description to the repository that best describes the package
4. Make the repository public for community suggestions and feedbacks
5. Initialize this repository with a README which best represents the package.

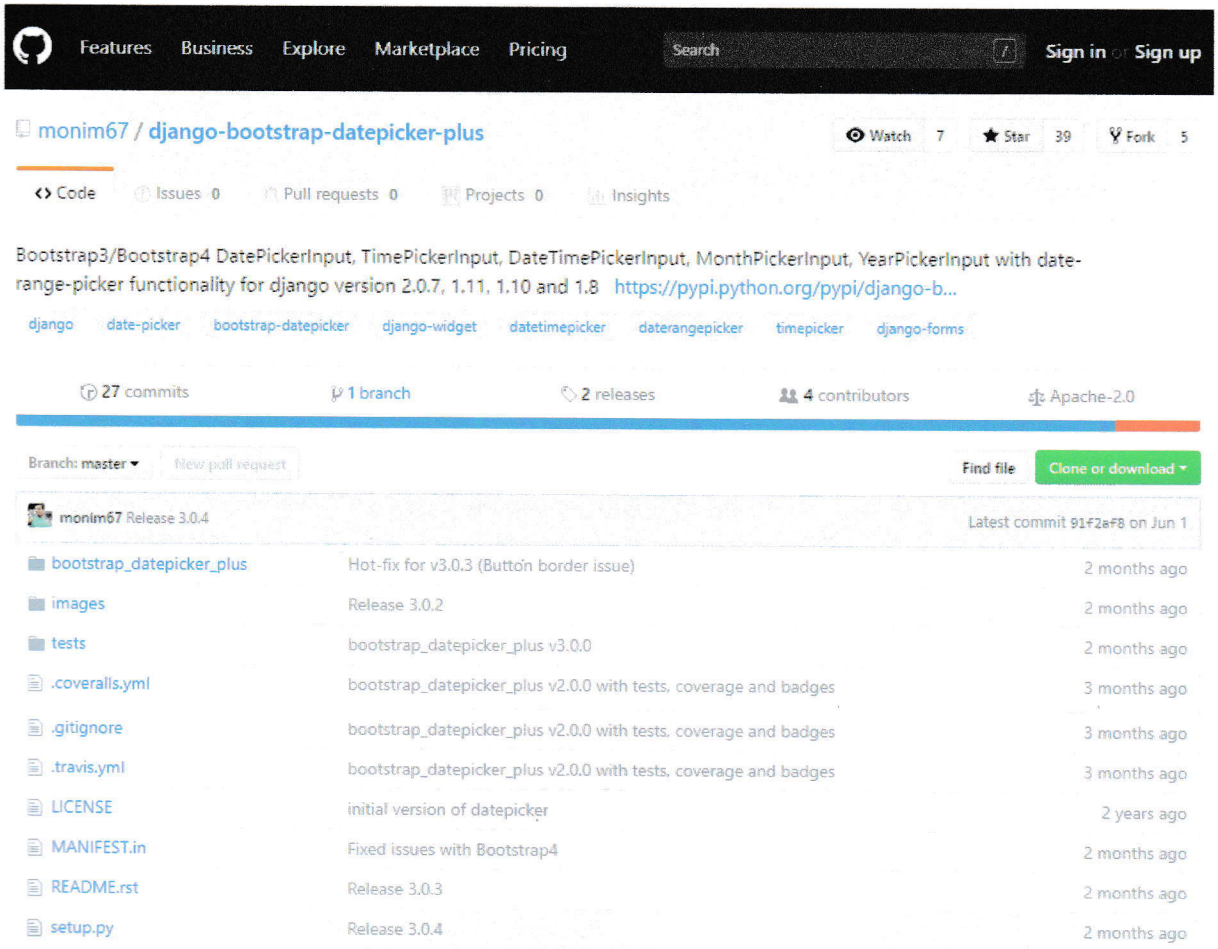


Figure 5.1: The GitHub page of the Package

5.4 Publishing to PyPI

The Python Package Index (PyPI) is a repository of software for the Python programming language. PyPI helps find and install software developed and shared by the Python community. Package authors use PyPI to distribute their software.

5.4.1 Objectives

1. Host the package on PyPI for easy installation via pip [2].
2. Give the package a unique name that is self-explanatory and search friendly
3. Upload and maintain different versions of the package with proper dependencies.

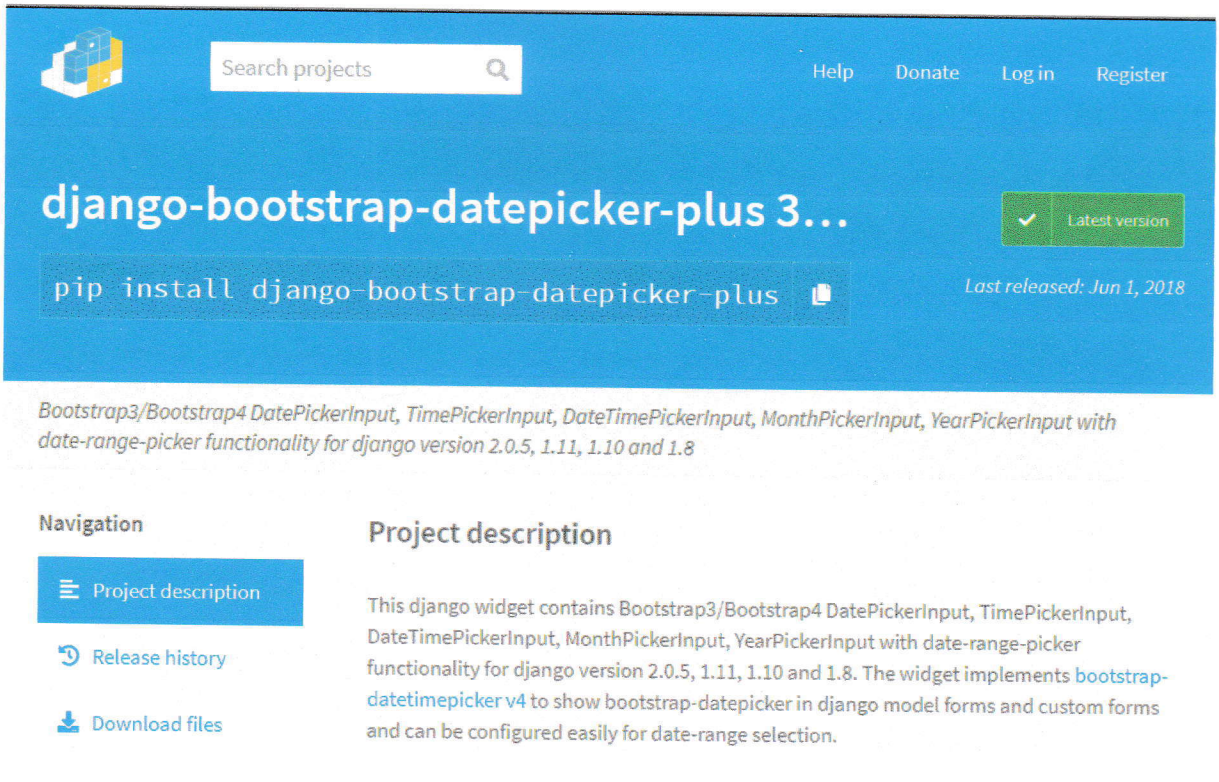


Figure 5.2: The PYPI page of the Package

5.5 Publishing to Django Packages

Django Packages is a directory of reusable apps, sites, tools, and more for your Django projects. This is the place to list python packages that are specific to django framework. This directory contains a huge collection of django reusable resources for the passionate developers to find and implement in their own projects. [3]

5.5.1 Objectives

1. List the package to a place specific to Django development
2. Make the package more available to Django developers.

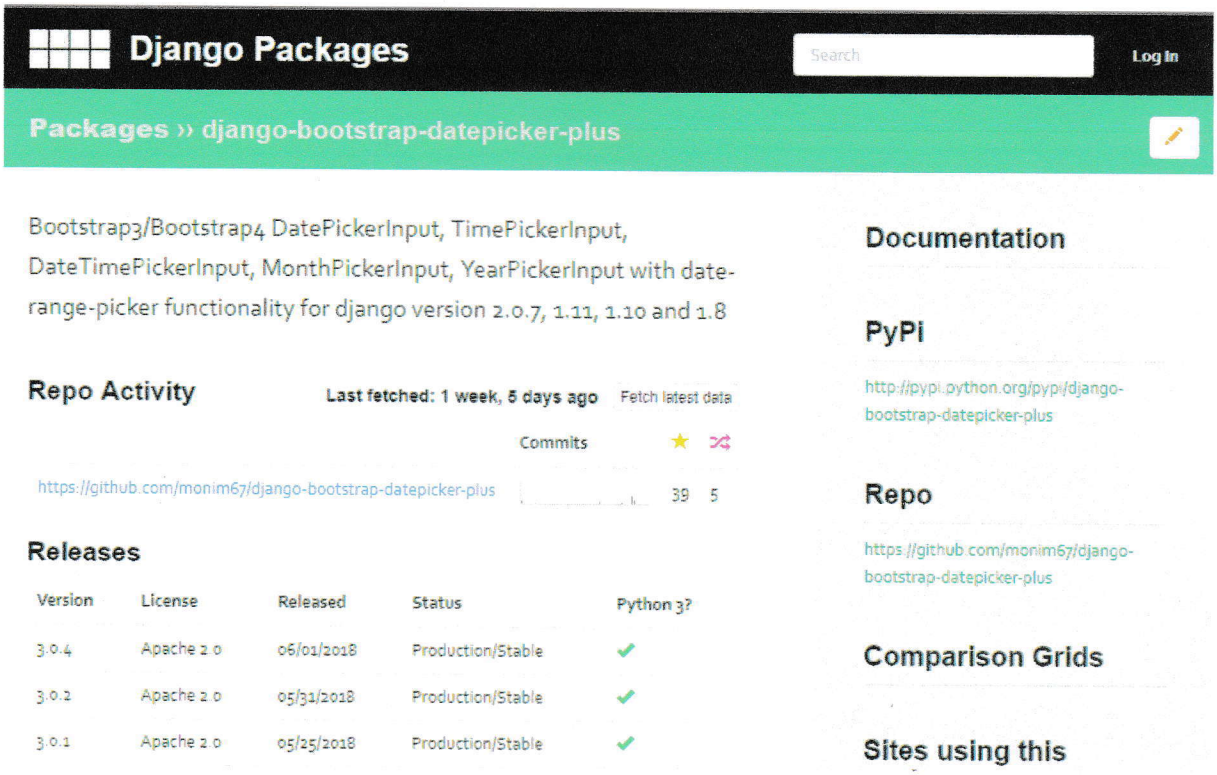


Figure 5.3: The Django Packages page of the Package

5.6 Publish Demo Pages

5.6.1 Demo with Bootstrap 3

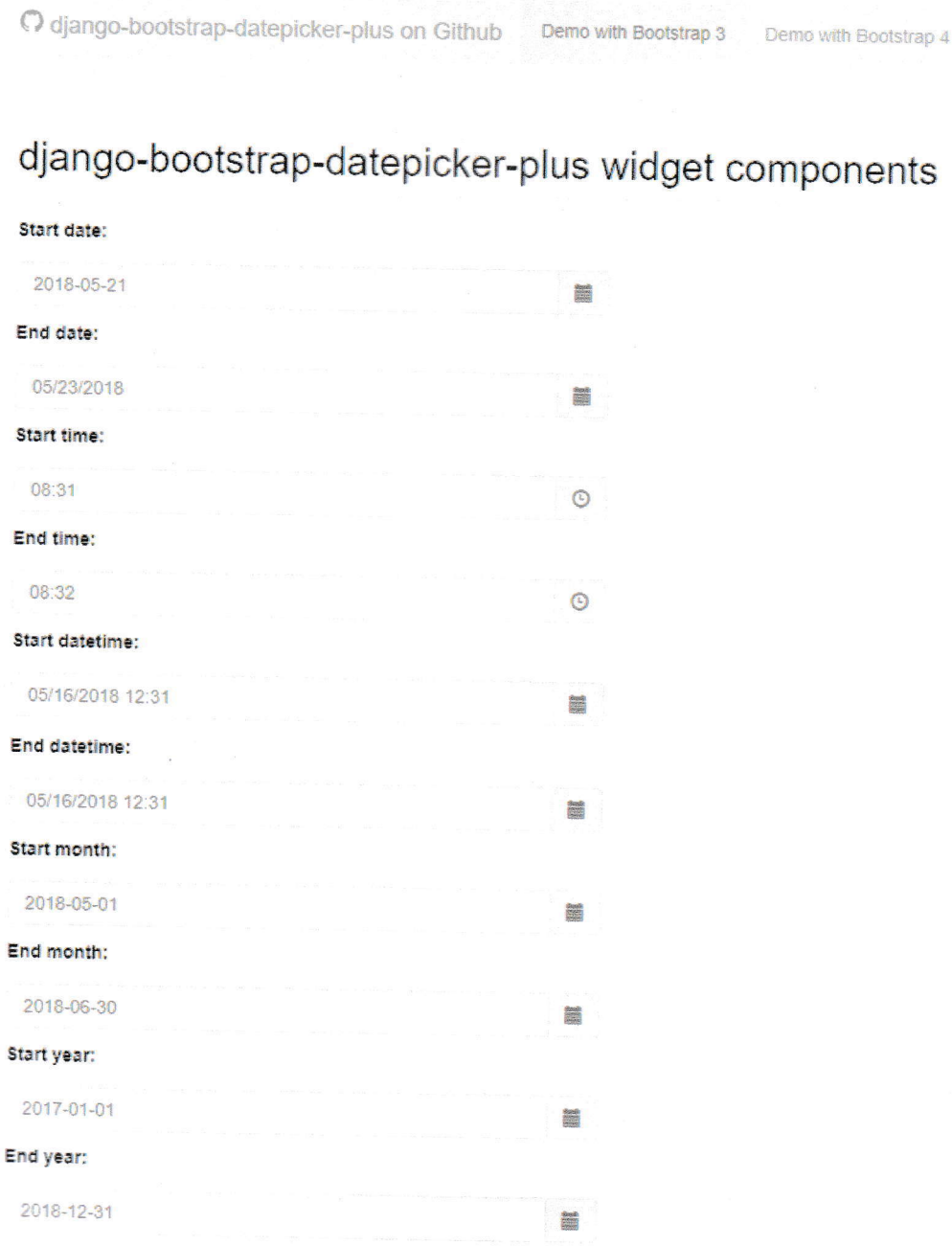


Figure 5.4: The demo page with bootstrap 3

5.6.2 Demo with Bootstrap 4

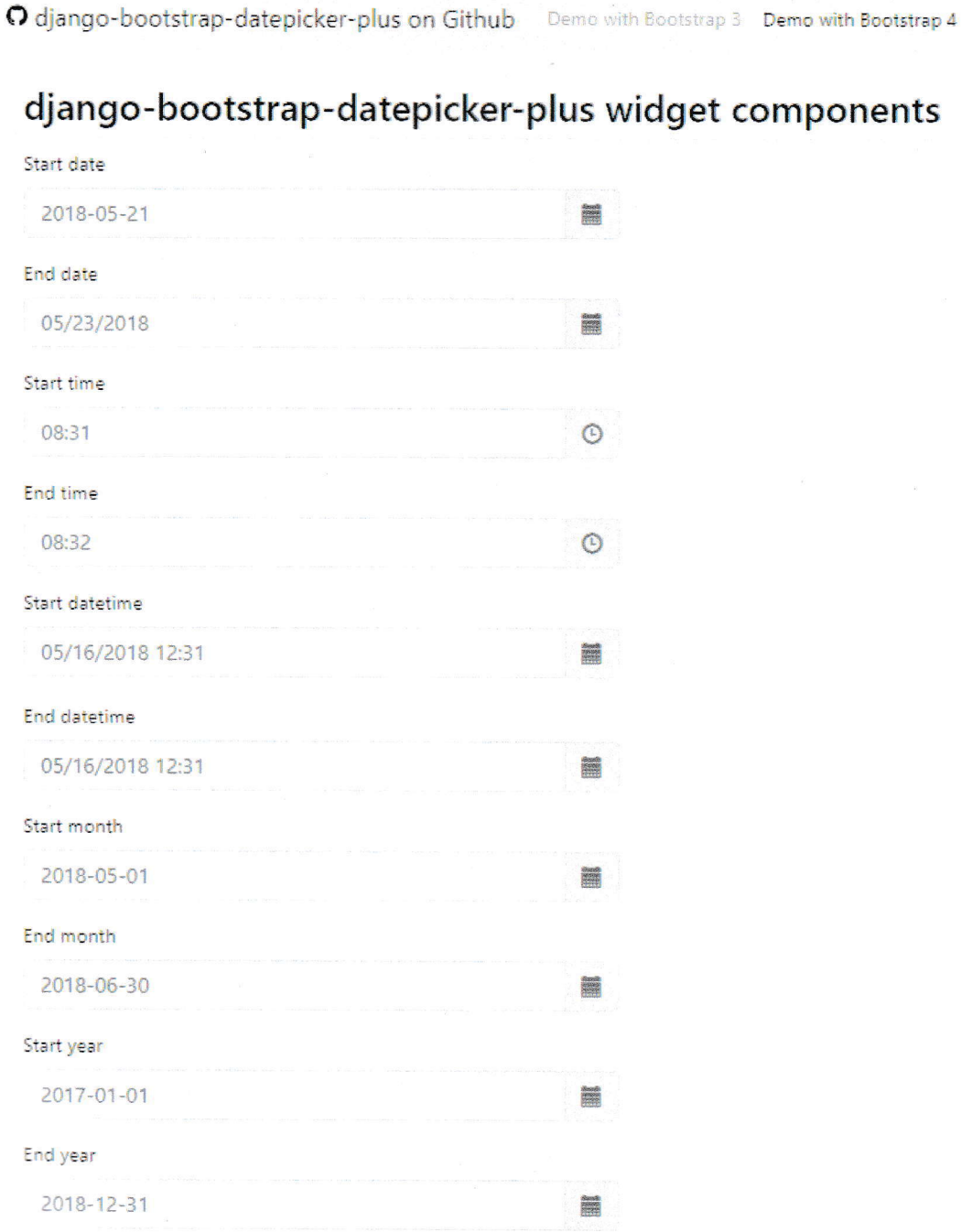


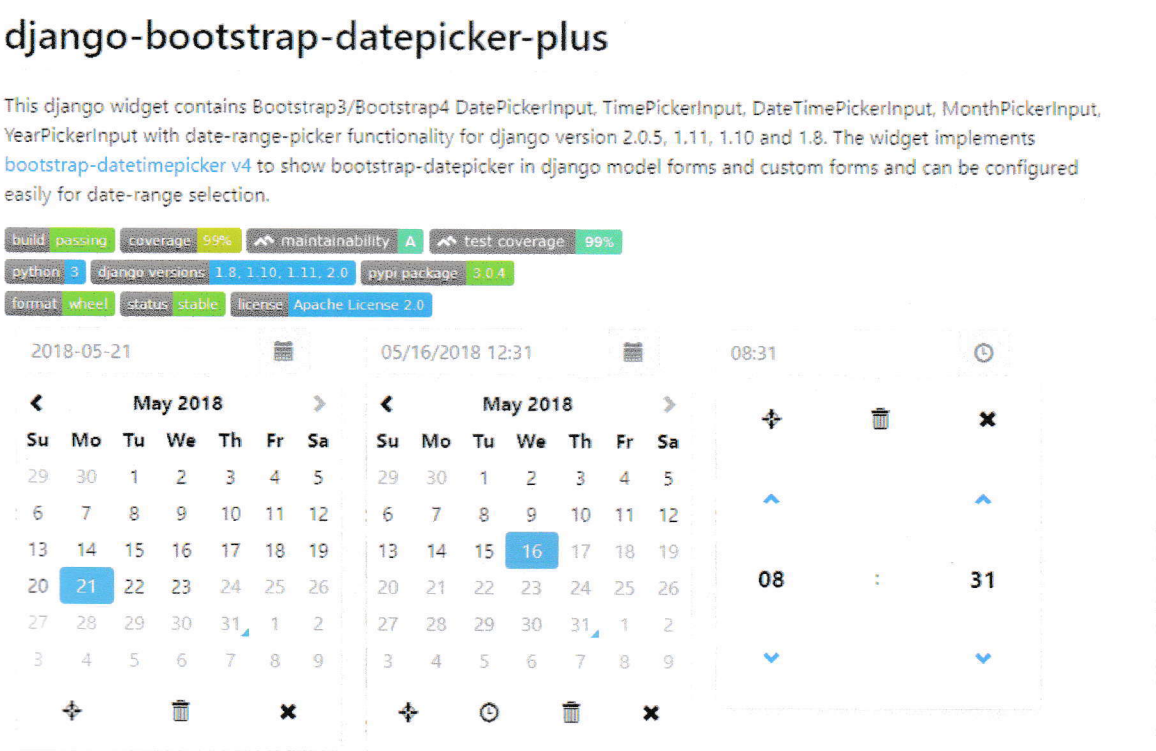
Figure 5.5: The demo page with bootstrap 4

5.7 Publish package documentation (README)

No matter how the quality of a product is, a book is always judged by it's cover. The cover of a repository is the README of a Github repository. The README should contain the following informations to reflect the important aspects of the package.

1. A small introduction to what the repository is all about.
2. Some repository badges to reflect some insights of the quality of code inside.
3. An image to to display the end result of the package.
4. Some demo links to see the package in action.
5. Prerequisites of the package.
6. Detailed usage guide of the package.
7. Licence and Acknowledgements

5.7.1 The Introduction



5.7.2 Demo, Getting Started

Demo

- With Bootstrap 3.
- With Bootstrap 4.

Getting Started

Prerequisites

- Python >= 3.3
- Django >= 1.8
- Bootstrap >= 3
- jquery >= 1.7.1

Installing

Install the widget via pip

```
pip install django-bootstrap-datepicker-plus
```

Add `bootstrap_datepicker_plus` to the list of `INSTALLED_APPS` in your `settings.py` file.

```
INSTALLED_APPS = [
    # Add the following
    'bootstrap_datepicker_plus',
]
```

`jQuery` is needed for `datepicker` to render, make sure you have jQuery in your template, or you can use `jQuery` included with `Bootstrap` by setting `include_jquery` option to `True` in your `settings.py` file. If you don't have `BOOTSTRAP3 / BOOTSTRAP4` settings block you have to create one.

```
# Use BOOTSTRAP3 if you are using Bootstrap 3
BOOTSTRAP4 = {
    'include_jquery': True,
}
```

Make sure you have bootstrap tags in your template along with `forms.media` tag, it adds all JS and CSS resources needed to render the date-picker.

```
{% load bootstrap4 %}       {# import bootstrap4/bootstrap3 #}
{% bootstrap_css %}         {# Embed Bootstrap CSS #}
{% bootstrap_javascript jquery='full' %}  {# Embed Bootstrap JS+jQuery #}

{% block extrahead %}       {# Extra Resources Start #}
{{ form.media }}            {# Form required JS and CSS #}
{% endblock %}              {# Extra Resources End #}
```

5.7.3 Usage

Usage

Custom Form usage

```
# File: forms.py
from bootstrap_datepicker_plus import DatePickerInput
from django import forms

class ToDoForm(forms.Form):
    todo = forms.CharField(
        widget=forms.TextInput(attrs={"class": "form-control"})
    )
    date = forms.DateField(
        widget=DatePickerInput(format='%m/%d/%Y')
    )
```

Model Form usage

```
# File: forms.py
from bootstrap_datepicker_plus import DatePickerInput
from django import forms

class EventForm(forms.ModelForm):
    class Meta:
        model = Event
        fields = ['name', 'start_date', 'end_date']
        widgets = {
            'start_date': DatePickerInput(), # default date-format %m/%d/%Y will be used
            'end_date': DatePickerInput(format='%Y-%m-%d'), # specify date-frmat
        }
```

Types of DatePickers

The widget contains all types of date-picker you may ever need.

```
# File: forms.py
from bootstrap_datepicker_plus import DatePickerInput, TimePickerInput, DateTimePickerInput, MonthPickerInput, YearPic
from django import forms

class EventForm(forms.ModelForm):
    class Meta:
        model = Event
        fields = ['start_date', 'start_time', 'start_datetime', 'start_month', 'start_year']
        widgets = {
            'start_date': DatePickerInput(),
            'start_time': TimePickerInput(),
            'start_datetime': DateTimePickerInput(),
            'start_month': MonthPickerInput(),
            'start_year': YearPickerInput(),
        }
```

5.7.4 Customising the Options

Customize the Options

The DatePicker can be customised by passing options to it. The `options` will be passed to the JavaScript datepicker instance, and are documented and demonstrated in Bootstrap Datepicker Options Reference.

```
# File: forms.py
from bootstrap_datepicker_plus import DatePickerInput
from django import forms

class EventForm(forms.ModelForm):
    class Meta:
        model = Event
        fields = ['name', 'start_date', 'end_date']
        widgets = {
            'start_date': DatePickerInput(format='%m/%d%Y'), # python date-time format
            'end_date': DatePickerInput(
                options={
                    "format": "MM/DD/YYYY", # moment date-time format
                    "showClose": True,
                    "showClear": True,
                    "showTodayButton": True,
                }
            ),
        }
```

Note: You can specify the date-time format by passing a python date-time format as format parameter (see start_date in the example), or by passing a moment date-time format as an option (see end_date in the example). If both are specified then the moment format in options will take precedence.

5.7.5 Licence and Acknowledgements

License

This project is licensed under Apache License 2.0 - see the LICENSE file for details.

Acknowledgments

This project implements Eonasdan/bootstrap-datetimepicker to display date-pickers. The project was initially forked from pbucher/django-bootstrap-datepicker.

© 2018 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub API Training Shop Blog About

Chapter 06

Testing & Quality Management

6.1 Unit-Testing

In computer programming, unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use.

6.1.1 Unit-Testing tools

There are many tools for creating tests in Python. Some of these tools, such as pytest, replace the built-in unittest framework. Other tools, such as nose, are extensions that ease test case creation. Note that many of these tools are also used for integration testing by writing the test cases to exercise multiple parts of code at once.

- **unittest** is the built-in standard library tool for testing Python code.
- **pytest** is a complete testing tool that emphasizes backwards-compatibility and minimizing boilerplate code.
- **nose** is an extension to unittest that makes it easier to create and execute test cases.
- **Hypothesis** is a unit test-generation tool that assists developers in creating tests that exercise edge cases in code blocks. The best way to get started using Hypothesis is by going through the well-written quickstart.
- **testify** was a testing framework meant to replace the common unittest+nose combination. However, the team behind testify is transitioning to pytest so it's recommended you do not use testify for new projects.

6.1.2 Code for Testing

```
import unittest

import django
from django.conf import settings
settings.configure()
django.setup()

from bootstrap_datepicker_plus import DatePickerInput,
                                      TimePickerInput,
                                      DateTimePickerInput,
                                      MonthPickerInput,
                                      YearPickerInput

from bootstrap_datepicker_plus.DatePickerBaseInput import DatePickerDictionary


Run Test | Debug Test
class TestDatePicker(unittest.TestCase):

    def setUp(self):
        pass

    Run Test | Debug Test
    def test_format_py2js(self):
        self.assertEqual(DatePickerInput.format_py2js(
            '%Y-%m-%d'), 'YYYY-MM-DD')
        self.assertEqual(DatePickerInput.format_py2js(
            '%d/%m/%Y'), 'DD/MM/YYYY')

    Run Test | Debug Test
    def test_format_js2py(self):
        self.assertEqual(DatePickerInput.format_js2py(
            'YYYY-MM-DD'), '%Y-%m-%d')
        self.assertEqual(DatePickerInput.format_js2py(
            'DD/MM/YYYY'), '%d/%m/%Y')

    Run Test | Debug Test
    def test_year(self):
        yearpicker1 = YearPickerInput().start_of('tes1')
        yearpicker2 = YearPickerInput().end_of('tes1')
        self.assertEqual(yearpicker1.config['options']['format'], 'YYYY-01-01')
        self.assertEqual(yearpicker2.config['options']['format'], 'YYYY-12-31')

    Run Test | Debug Test
    def test_datepicker(self):
        startpicker = DatePickerInput().start_of('test')
        endpicker = DatePickerInput(
            options={'format': "YYYY-MM-DD"}).end_of('test')
        self.assertRaises(KeyError, lambda: endpicker.end_of('undefined'))
        context = startpicker.get_context("input_name", '2018-04-12', {})
        self.assertTrue(len(context['widget']['attrs']['dp_config']) > 0)

    Run Test | Debug Test
    def test_backcompatibility(self):
        DatePickerInputEx = DatePickerDictionary.get_base_input(True)
        dummypicker = DatePickerInputEx()
        self.assertEqual(dummypicker.format_value(''), None)
        # context = dummypicker.get_context("input_name", '2018-04-12', {})
        # self.assertEqual(context['widget']['attrs']['name'], 'input_name')
        html = dummypicker.render("input_name", '2018-04-12', {})
        self.assertTrue(len(html) > 0)
```

6.1.3 Unit-Testing with pytest

pytest is a mature full-featured Python testing tool that helps you write better programs. The pytest framework makes it easy to write small tests, yet scales to support complex functional testing for applications and libraries.

Test Results:

```
> Executing task: npm run test <


> django-bootstrap-datepicker-plus@3.0.4 test d:\Python\monim67\django-bootstrap-datepicker-plus
> pytest --cov=bootstrap_datepicker_plus tests

============================= test session starts =============================
platform win32 -- Python 3.6.3, pytest-3.4.0, py-1.5.2, pluggy-0.6.0
rootdir: d:\Python\monim67\django-bootstrap-datepicker-plus, inifile:
plugins: cov-2.5.1
collected 5 items

tests\test_widgets.py .....                                              [100%]

---------- coverage: platform win32, python 3.6.3-final-0 -----------
Name                                                 Stmts   Miss  Cover
------------------------------------------------------------------------
bootstrap_datepicker_plus\DatePickerBaseInput.py        75      0   100%
bootstrap_datepicker_plus\DateTimeBaseInputEx.py        39      1    97%
bootstrap_datepicker_plus\__init__.py                    5      0   100%
bootstrap_datepicker_plus\widgets.py                    24      0   100%
------------------------------------------------------------------------
TOTAL                                                  143      1    99%


========================== 5 passed in 1.59 seconds ===========================

Terminal will be reused by tasks, press any key to close it.
```

Figure 6.1: pytest unit-test results

6.1.4 Unit-Testing with Travis-CI

Travis CI is a hosted, distributed continuous integration service used to build and test software projects hosted at GitHub. Open source projects may be tested at no charge via travis-ci.org.

6.1.4.1 Travis-CI build flow



6.1.4.2 Travis-CI Result

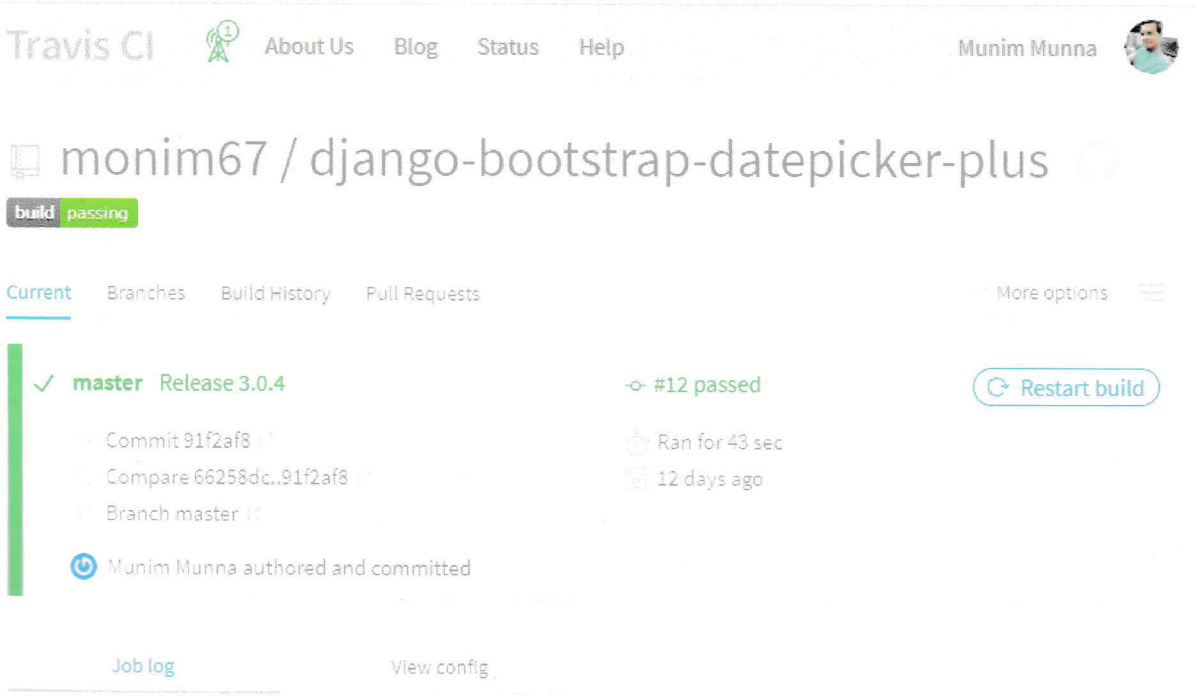


Figure 6.2: travic-CI build result

6.1.4.3 Travis-CI build history

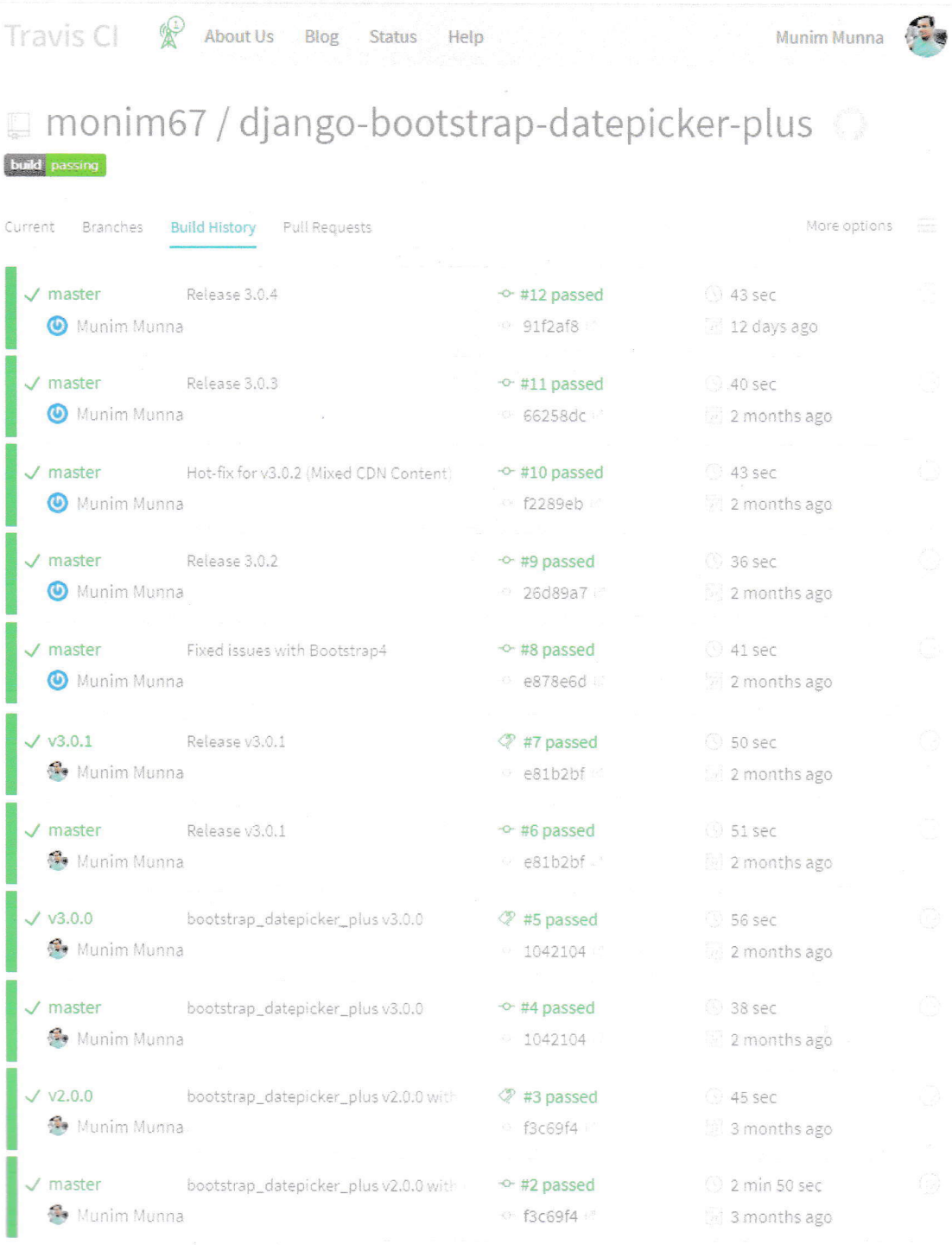
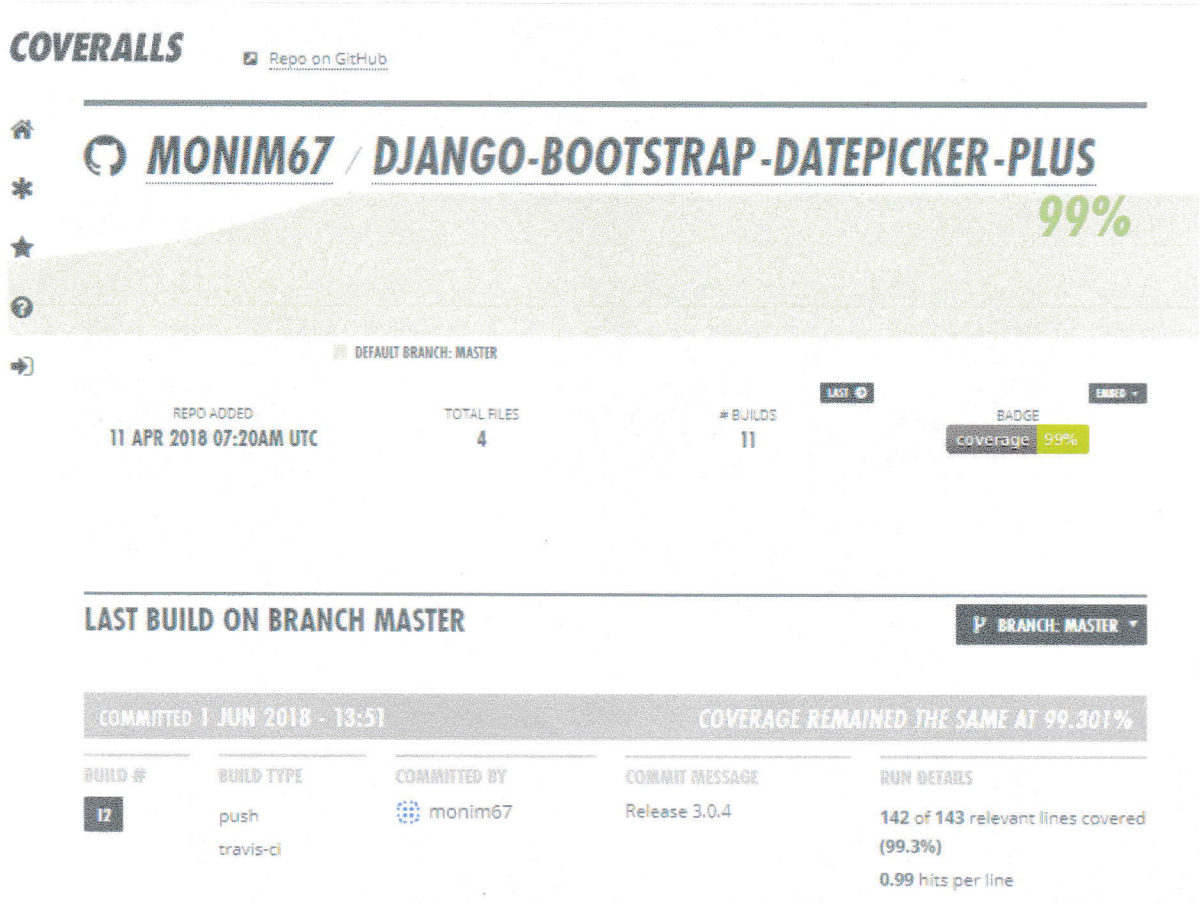


Figure 6.3: travic-CI build log

6.2 Code-Coverage

In computer science, test coverage is a measure used to describe the degree to which the source code of a program is executed when a particular test suite runs. A program with high test coverage, measured as a percentage, has had more of its source code executed during testing, which suggests it has a lower chance of containing undetected software bugs compared to a program with low test coverage.

6.2.1 Coveralls.io stats



6.2.2 Coveralls.io details and builds

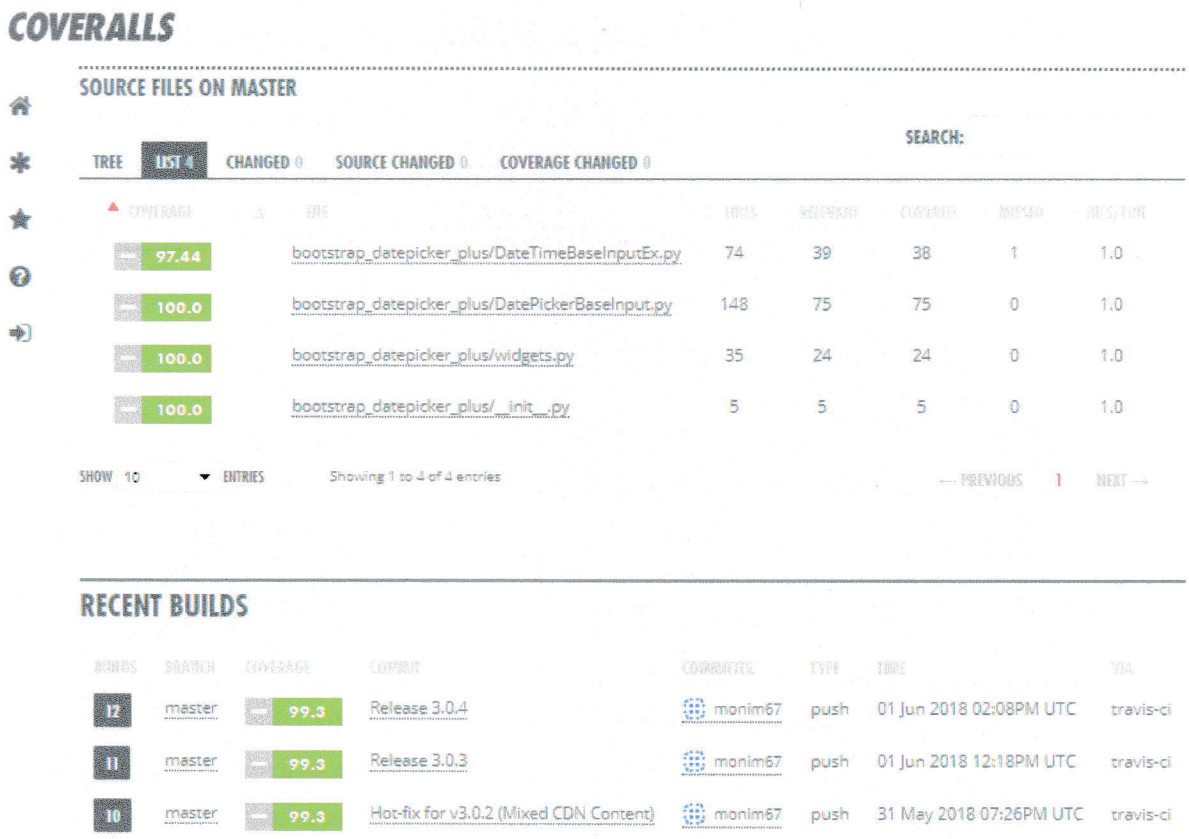


Figure 6.4: coveralls.io report

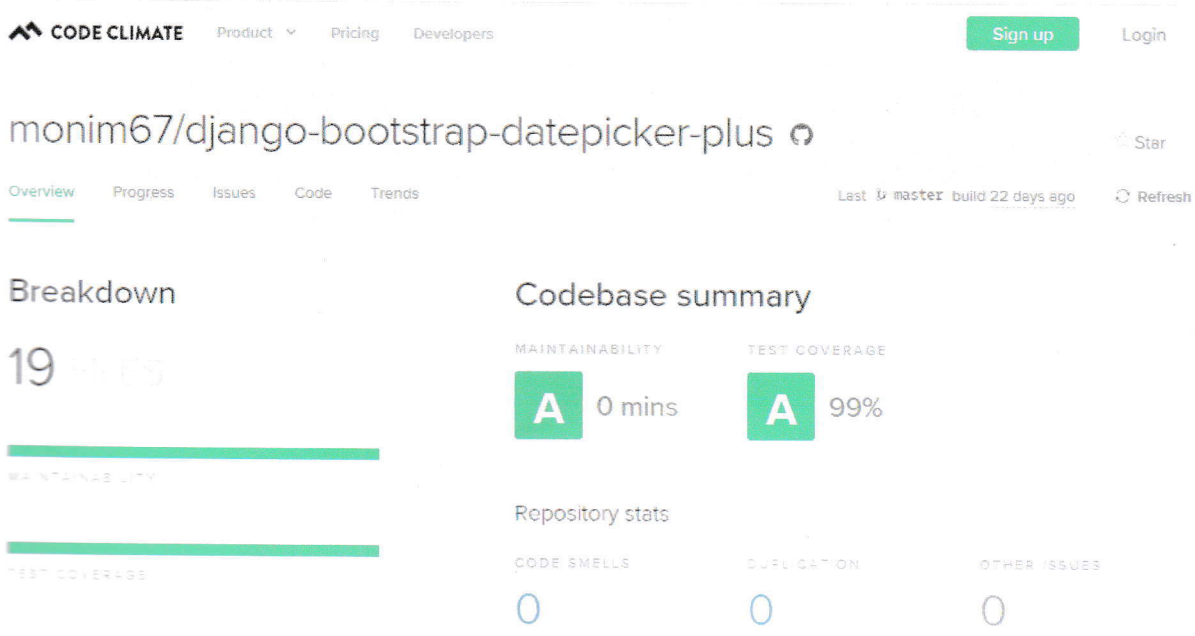
6.3 Maintainability

Maintainability is an estimate of technical debt in the repo based on a standardized 10-point assessment of Duplication, Cyclomatic Complexity, Cognitive Complexity, and structural issues.

6.3.1 Code-Climate Report

The 10 checks outlined below are enabled by default in Code Climate.

- **Argument count** - Methods or functions defined with a high number of arguments
- **Complex logic** - Boolean logic that may be hard to understand
- **File length** - Excessive lines of code within a single file
- **Identical blocks of code** - Duplicate code which is syntactically identical (but may be formatted differently)
- **Method complexity** - Functions or methods that may be hard to understand (Cognitive Complexity)
- **Method count** - Classes defined with a high number of functions or methods
- **Method length** - Excessive lines of code within a single function or method
- **Nested control flow** - Deeply nested control structures like if or case
- **Return statements** - Functions or methods with a high number of return statements
- **Similar blocks of code** - Duplicate code which is not identical but shares the same structure (e.g. variable names may differ)

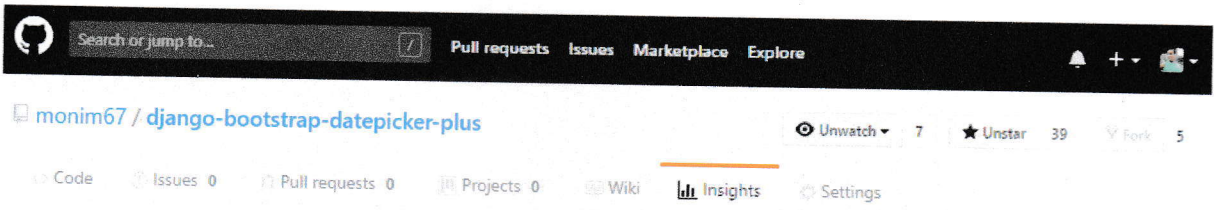


Chapter 07

Project Success

7.1 Project Stats on Github

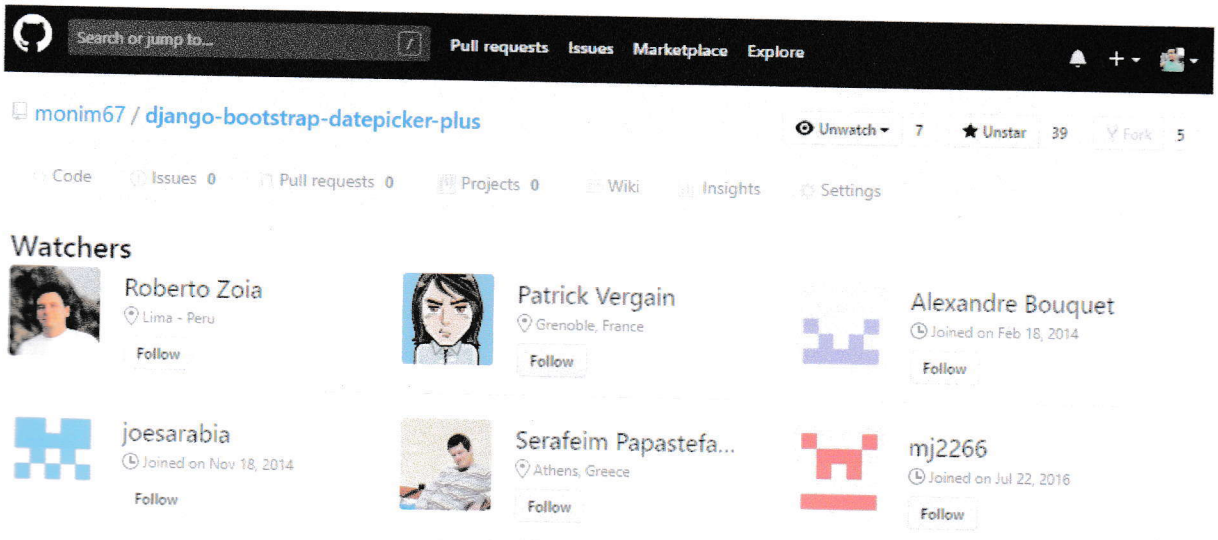
- Watchers: 7
- Stars: 39
- Forks: 5



7.1.1 Watchers

Watching a Repository registers the user to receive notifications on new discussions, as well as events in the user's activity feed. See Repository Starring for simple repository bookmarks.

These are the developers currently watching this repository for instant notification of any updates published to this repository.

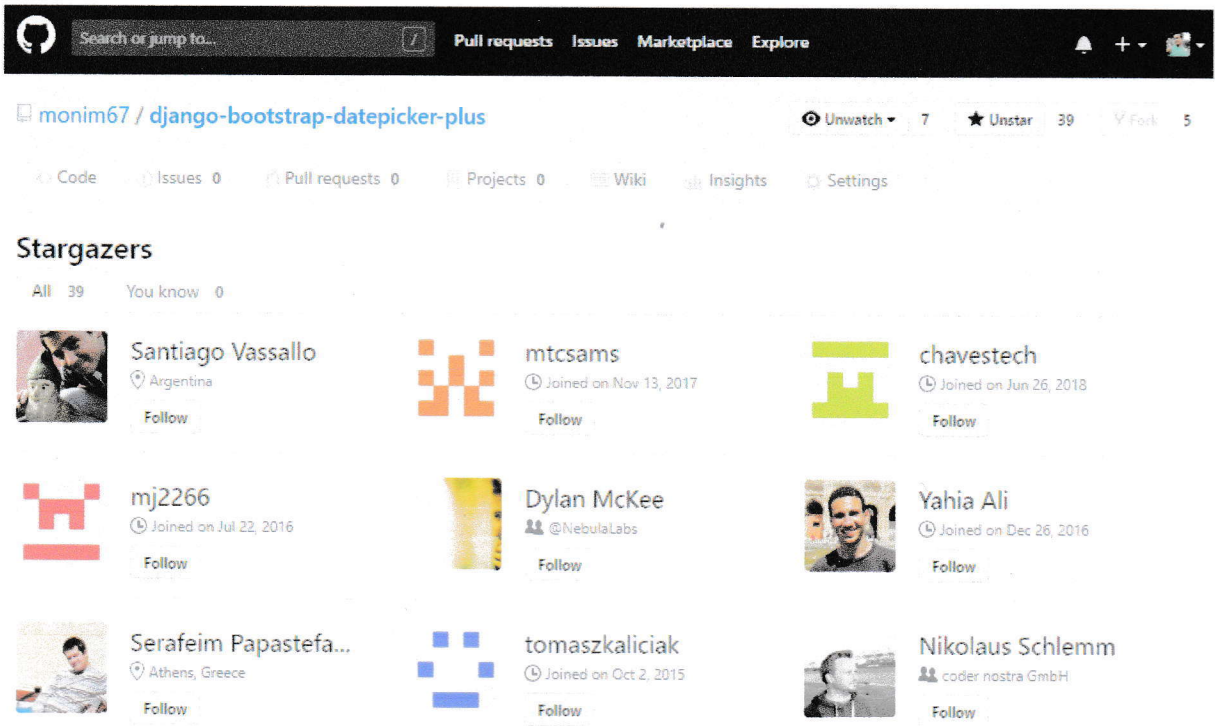


7.1.2 Stars

Repository Starring is a feature that lets users bookmark repositories. Stars are shown next to repositories to show an approximate level of interest. Stars have no effect on notifications or the activity feed. For that, see Repository Watching.

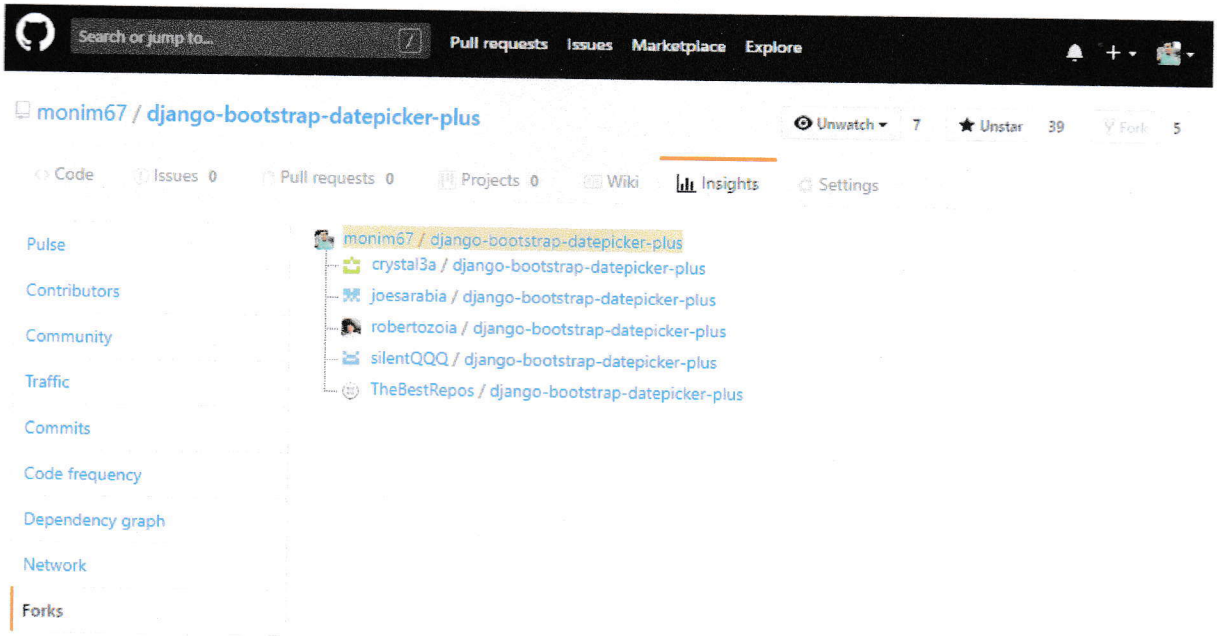
The project has received 39 stars and this number is still growing as more and more developers find out this package. The usefulness and quality of a project is reflected by these stars provided by the developers who are using it.

Starred by

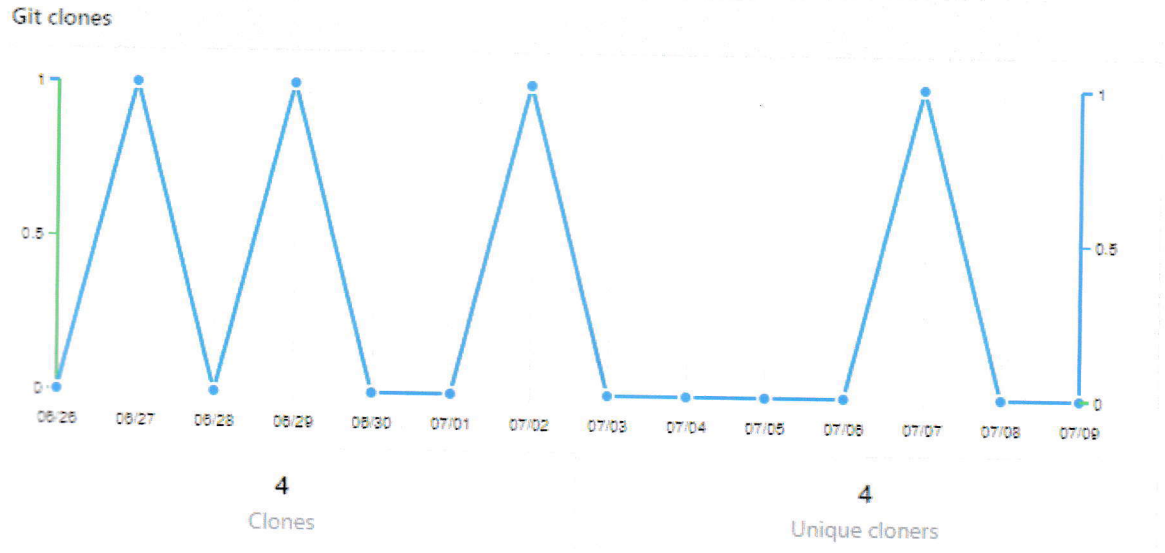


7.1.3 Forks

A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. Most commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea.

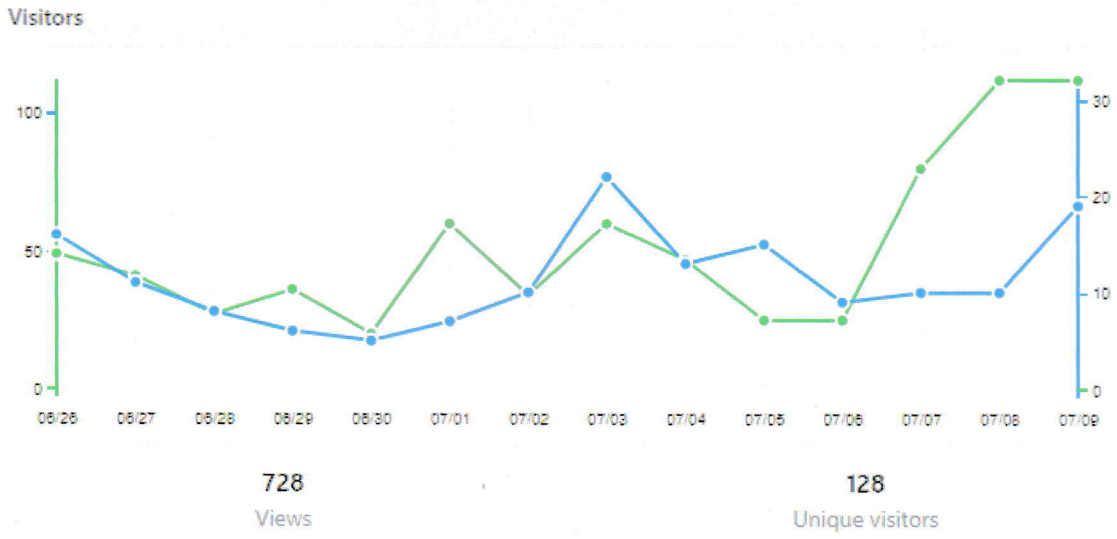


7.1.3.1 Clone Graph



7.1.4 Visitor Graph

The visitor count keeps on rising as days pass by proving the project is reaching more and more developers making this project a success.



Referring sites

Site	Views	Unique visitors
Google	254	66
reddit.com	69	7
github.com	50	12
monim67.github.io	46	7
djangopackages.org	33	12
pypi.org	13	4
pydigger.com	2	1
ctolib.com	1	1
cn.bing.com	1	1

Popular content

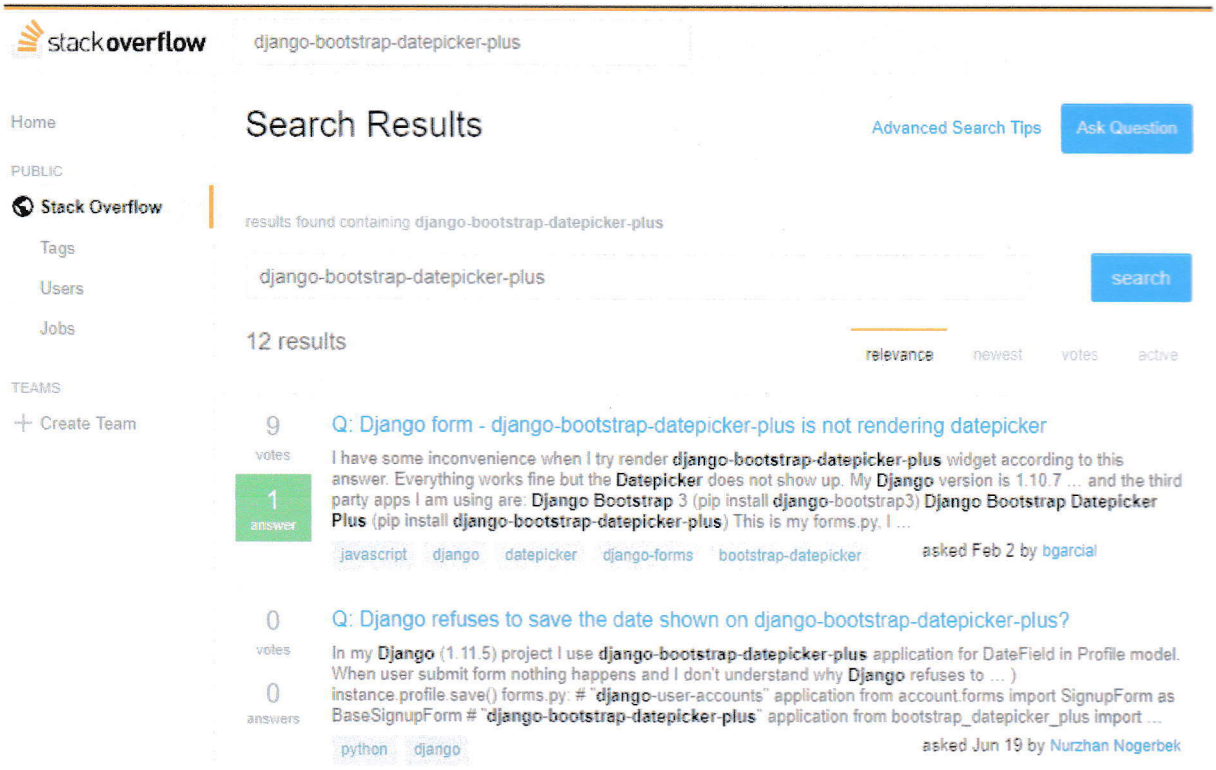
Content	Views	Unique visitors
monim67/django-bootstrap...	340	141
django-bootstrap-datepicke...	71	31
django-bootstrap-datepicke...	40	18
django-bootstrap-datepicke...	33	13
django-bootstrap-datepicke...	22	16
django-bootstrap-datepicke...	21	17
django-bootstrap-datepicke...	17	9
django-bootstrap-datepicke...	16	12
django-bootstrap-datepicke...	15	7
django-bootstrap-datepicke...	13	11

7.2 Activity on Stack-Overflow

7.2.1 Questions on Stack-Overflow

There are 12 questions on stack-overflow about this project all asked by django developers who were having difficulties to implement this package to their django powered site. These questions provided a lot information to help make the project more user-friendly and easier to implement.

This activity on stack-overflow shows this package is receiving traction more and more developers are being interested in using it in their own project.



7.2.2 Answers on Stack-Overflow

This answer were posted encouraging the use of this package in django site describing the installation setup and usage of it which has received 5 votes by the community members. This shows developers are really appreciating this package as a useful tool for their django sites.

5

When it comes to date-pickers my choice is Bootstrap Datepicker. You can implement it in your django app by using django-bootstrap-datepicker-plus which works both on newer and older DJango versions. I maintain the repository and tested it working in DJango version *1.8*, *1.10*, *1.11* and *2.0.4*.



The setup is quite easy. You just install it.

```
pip install django-bootstrap-datepicker-plus
```

Import the widget in your **forms.py** file

```
from bootstrap_datepicker_plus import DatePickerInput
```

Chapter 08

Conclusion

8.1 The Experience

Working in this project was a big opportunity for me. I have learned a lot about the open-source contribution to community software development. The following will indicate the learning from this project.

1. The design and development strategy of a django package.
2. Publishing a well-appreciated project to the biggest online community Github and maintain a meaningful commit history.
3. Unit-testing complex scenarios of software development.
4. Writing code with well accepted code writing guidelines (PEP8).
5. Working with the Travis-CI build system and incorporating build system to a repository.
6. Generate repository coverage reports with Coveralls.io.
7. Learning code maintenance guidelines provided by CodeClimate.
8. Writing clear and thorough documentation of how to use a software.
9. Packaging and uploading a package to PYPI Python Packages Index.
10. Developing a project with end-user follow ups, bug reports and feature requests.
11. Developing a product with a thoughtful user-experience.

8.2 Future Work

The package is published online on both Github and PyPI package index and thus available to django developers to implement in their django powered sites. It's receiving lot of appreciation and new features suggestions that will keep the project growing. Here is a summary of my present thoughts of future works.

1. Developing a clock-like UI for the time-picker that looks more neat and provide easier selection of time.
2. There are some bugs present in the date-picker library used to render the UI all listed on the issues page of Github, need to address those.
3. Make the project compatible to the latest django version when django version 2.1 is released which is now in beta phase.

8.3 Conclusion

This project is a work of lots of late night works involving a lot of patience debugging for different versions of django both latest and older to keep it suitable for a wide range of django developers. Without the suggestions, follow-ups and thumbs ups of the reddit and github community it would not have been possible at all. It would be my great pleasure to continue to work on it to keep it alive for the upcoming django releases and serve the open-source community of awesome developers.

8.4 References

[1] Github repository: https://github.com/monim67/django-bootstrap-datepicker-plus

[2] PyPI package: https://pypi.org/project/django-bootstrap-datepicker-plus/

[3] Django packages index: https://djangopackages.org/packages/p/django-bootstrap-datepicker-plus/

[4] Bootstrap 3 Demo: https://monim67.github.io/django-bootstrap-datepicker-plus/

[5] Bootstrap 4 Demo: https://monim67.github.io/django-bootstrap-datepicker-plus/

[6] Travis-CI: https://travis-ci.org/monim67/django-bootstrap-datepicker-plus

[7] Coveralls.io: https://coveralls.io/github/monim67/django-bootstrap-datepicker-plus?branch=master

[8] CodeClimate: https://codeclimate.com/github/monim67/django-bootstrap-datepicker-plus/maintainability

[9] Reddit post: https://www.reddit.com/r/django/comments/8gza32/djangobootstrap_datepickerplus_django_widget_for/

[10] https://stackoverflow.com/questions/3367091/whats-the-cleanest-simplest-to-get-running-datepicker-in-django

[11] https://stackoverflow.com/questions/28148125/how-to-use-the-bootstrap-datepicker-in-django-app