Django Web Development Framework: Powering the Modern Web

Songtao Chen[1], Shahed Ahmmed[2], Karu Lal[3*], Chunhua Deming[4]

[1]Jiujiang Vocational and Technical College, Jiujiang, Jiangxi, **CHINA**
[2]Lecturer, Department of Business Administration, Fareast International University, Dhaka, **BANGLADESH**
[3]Integration Engineer, Ohio National Financial Services, **USA**
[4]National University of Singapore, **SINGAPORE**

[*]E-mail for correspondence: karu.lal84@gmail.com

ABSTRACT

Django is a web development framework that is both powerful and flexible, and it has become an essential component in developing modern web applications. This open-source Python framework is lauded for its pragmatic design, precise code, and extensive collection of built-in features that speed up the software development process. Django's foundation is built on the "Don't Repeat Yourself" (DRY) principle, which streamlines the development of complex web applications by reducing the required duplication. Its Model-View-Controller (MVC) architectural pattern enables a clear separation of concerns, simplifying both the process of creation and the maintenance of the system. Object-relational mapping, or ORM for short, is a mechanism that Django uses to simplify and streamline database interactions by doing away with the need to perform complicated SQL queries. This review attempt will serve as a jumping-off point for our Django journey. This study will help to get started with Django by offering an overview of the fundamental principles and processes to begin building web apps.

Keywords: Django, Python, Authentication, Scalability, Modern Web Applications, DRY, Security, Reusable Components

INTRODUCTION

Django is a web application framework that stands out in the constantly shifting world of online development as a robust and dependable solution for creating modern web applications. Django is an open-source web development framework that has achieved a great deal of popularity due to its capacity to simplify the development process, keep the code clean and effective, and enable developers to construct web applications that are dynamic, safe, and scalable (Thaduri et al., 2016). Django is a framework built on the Python programming language and abides by the "Don't Repeat Yourself" (DRY) concept at its core. This indicates that it encourages developers to avoid writing redundant code and instead build code that is concise and easy to maintain (Dekkati et al., 2019). It does this, which results in a significant acceleration of the development cycle and ease in the maintenance load. As a result, it is an appropriate solution for projects of any size. Django's architectural pattern is called Model-View-Controller (MVC), ensuring that concerns are kept separate. This separation improves the organization of the code as well as the maintainability of the code, allowing developers to concentrate on specific components of their web applications (Deming et al., 2018). Because the framework's object-relational mapping (ORM) mechanism encapsulates database interactions, the requirement to write complicated SQL queries is eliminated. This not only makes the management of databases easier but also encourages a more intuitive approach to data handling.

One of the most notable aspects of Django is the extensive collection of tools and libraries that come pre-installed with the framework. These tools cover many facets of web development, such as user authentication, various security measures, and an extensive library of reusable components known as "apps" (Dekkati et al., 2016). Because these resources are readily available, developers don't need to keep spinning the wheel for each new project; instead, they can focus on adding unique functionality to the

application. Scalability is another area in which Django performs quite well. Because it provides a variety of deployment options, it enables web applications to expand without causing any disruptions to their functioning and the flow of traffic. Django's scalability means that your application may quickly broaden in scope, regardless of whether you are developing a straightforward blog or a sophisticated e-commerce platform. Django is still one of the most popular frameworks for modern online development. It is robust, dependable, and flexible, allowing developers to tap into the web's full potential. This book will delve into the intricacies of Django, studying its features, best practices, and the numerous ways it continues to power the modern web (Cao et al., 2013).

GETTING STARTED WITH DJANGO

Because of its effectiveness, scalability, and comprehensive toolset, Django's open-source web development framework, built on Python, has garnered significant popularity in recent years. Additionally, Django provides an authentication system, security features, and a vast library of reusable components known as "apps," which enables developers to concentrate on application-specific logic. In addition, it offers several different deployment methods and embraces best practices for scalability (Dekkati & Thaduri, 2017). Django has become the platform of choice for web developers since it enables them to construct dynamic, safe, and effective web applications that are suitable for the requirements of the current web landscape.

Installation: Installing Django is the first thing you must do to begin working with it. To install Django, you can use pip, Python's package manager. Open your computer's command prompt or terminal and type the following command into it:

pip install django

This will begin downloading the most recent Django version and installing it on our computer.

Creating a Django Project: The following command must be entered to initiate the creation of a new project once Django has been successfully installed:

django-admin startproject projectname

Change "project name" to reflect the name of your undertaking. With this command, a directory that has all of the required project files and structure will be created.

Understanding the Project Structure: A particular project structure is adhered to by Django, and this structure encompasses settings, URL routing, and a variety of application components. The following is a list of the primary files and directories with which you will interact:

- **'settings.py':** It stores the project's configurations. These settings may include database configurations, security settings, and other project-specific parameters.
- **'urls.py':** Where the URL patterns for your application are defined. These URL patterns govern how requests are mapped to views.
- **'views.py':** Views are Python functions that handle HTTP requests and return HTTP responses. 'views.py' is located in the Views directory. This is the portion of your web application where the logic will be written.
- **'models.py':** Models are what specify the structure of the database that your application uses. The ORM that comes with Django makes managing databases much more accessible.
- **`templates/`:** The HTML templates that render web pages are stored under this directory's 'templates/' folder.
- **'static/':** Where static files such as CSS, JavaScript, and picture files are kept.

Creating Your First App: A single project in Django may encompass more than one app. The following command must be executed to make an application:

python manage.py startapp appname

Change "appname" to reflect the name of your application. An application is a self-contained component of your project that encompasses a particular aspect of its functionality.

Defining Models: Using Python classes, the data structures for your application are defined within the models.py file. These classes are then translated into database tables, which simplifies the process of working with the data associated with your application (Hillar, 2016).

Creating Views: You write Python functions in your app's 'views.py' file to handle HTTP requests, then save the file. These functions provide HTTP replies and determine how your web pages will be displayed on the user's browser.

URL Routing: You can set the URL patterns that will direct requests to the relevant views in your app within the 'urls.py' file that is part of your project. Django uses this information to determine which view to run in response to a URL.

Running the Development Server: The following command will allow you to begin the process of starting up the development server:

python manage.py runserver

This will start a local server, and you may view your Django project using a web browser by going to http://127.0.0.1:8000/. This will launch the server.

After completing these preliminary activities, you will be prepared to construct your web application using Django. As you make progress, you will investigate increasingly complex functionalities, such as deploying your application to a production server, processing forms, and

authenticating users, among other things (Chen et al., 2019). As you continue to work with this robust web development framework, you will find that the vast documentation provided by Django and the active community of developers it supports are invaluable tools (Desamsetti & Lal, 2019).

OBJECT-RELATIONAL MAPPING

Our Python code and the actual database are separated by a chasm that may be bridged thanks to the Object-Relational Mapping (ORM) system included in the Django web development framework. This robust ORM system is one of the most critical components of the Django web development framework. Using the Django Object Relational Mapper (ORM), programmers may interact with databases in a high-level, Pythonic manner. This makes the maintenance of data models much simpler and more streamlined. This post will investigate how the Django ORM facilitates creating and manipulating data models by providing developers with more control (Lopez et al., 2013).

- **Abstraction of Database Operations:** Django ORM is an object-relational mapper that hides the complexity of database operations. Instead of manually creating raw SQL queries, developers can work with databases using Python classes and functions. Because of this abstraction, not only is the code easier to read and maintain, but it also guarantees that the application will continue to be database-agnostic. This allows you to move between other database management systems quickly.
- **Defining Data Models:** Python classes are used to determine the various data models utilized by Django. These classes are representations of database tables, and each attribute of the class corresponds to a column in the table. You could construct a data model for blog postings if you designed an application to manage blogs. Here is an illustration of a straightforward definition of a data model:

```
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```

This code defines the 'BlogPost' data model and includes fields for the title, content, and creation date. Based on this model, Django will automatically construct the relevant database table.

- **Migrations:** We can change our data models utilizing Django's migration mechanism, a vital ORM feature. These changes may be made without having to edit the database schema manually. When you make modifications to your data models, such as adding a new field or updating an existing one, you can create and apply migrations to update the database schema in a manner that is appropriate for the changes (Hosen et al., 2019). This procedure is automated, and as a result, it guarantees that your database schema will always be in sync with the code.
- **Querying the Database:** The Django ORM offers a high-level application programming interface (API) for conducting database queries. Quickly retrieving, filtering, and manipulating data can be accomplished using methods and filters. For instance, if you want to retrieve all blog articles that have been created after a given date, you can use the query that is provided here:

```
recent_posts = BlogPost.objects.filter(created_at__gte=some_date)
```

This simplifies the SQL query to make it easier to read and implement in Python.

- **Integration with Views and Templates:** The Django ORM can integrate without hiccups with the views and templates. You can send data from the database to your views, which will render that data in the appropriate templates. The process of displaying database content on your web pages is made more accessible due to this comprehensive integration.
- **Security and Validation:** SQL injection prevention and data validation are two examples of the security measures included in the Django ORM. These capabilities ensure that all data transactions are both safe and dependable.

VIEWS AND TEMPLATES

When it comes to web development, having a user interface that has been carefully created is frequently the most critical factor in developing a practical and exciting web application. The widely used Django framework for web development offers a dependable mechanism for defining views and templates, crucial elements in creating a website's user interface. In this article, we'll investigate how views and templates in Django make it possible for developers to design web applications that are both user-friendly and aesthetically pleasing (Zambelli et al., 2013).

- **Views-Handling Requests and Logic:** Views are Python methods in Django responsible for handling HTTP requests and determining how the application replies to such requests. The responsibilities of views are the processing of user input, interaction with the database, and rendering of template data. Views are also responsible for generating dynamic HTML content. They act as a connection point between the user and the backend logic of the system.

  Take, for instance, the situation where a user requests a particular blog article to be written. This post would be retrieved from the database by a view function,

which would also handle any additional logic (such as user authentication) and create the proper template to display the blog post to the user.

Django includes various pre-built views, such as generic class-based views, that can make routine activities more accessible. In addition, developers can design their custom views to meet multiple requirements.

- **Templates - Structuring HTML:** Developers can determine the structure and style of web pages by utilizing templates, which are an essential component of the Django framework. In most cases, templates are HTML files and contain blank spaces intended to be filled in with dynamic content. The data from the views are used to fill in these placeholders, which results in a dynamic and personalized experience for the user.

  For instance, while showing a list of blog posts, a template can loop over the articles and insert each post's title, content, and publication date in the relevant positions within the HTML structure (Thaduri, 2017). This can be done while the template generates the blog post list.

  It is possible to develop web pages that are dynamic and responsive by using the template language that Django provides. This language's rich logic capabilities, which include conditionals and loops, make this possible. Additionally, it allows template inheritance, making it possible for developers to produce an application with a consistent layout.

- **Integrating Views and Templates:** Building a user interface in Django requires a fundamental understanding of the interplay between views and templates. Views are responsible for transferring data to templates, which are then rendered to produce HTML replies sent back to the user. This separation of concerns makes it possible to have a clean and well-organized codebase, making it more straightforward for developers to work together and maintain the program over time (Ravindran, 2015).

  Django includes a function called render that may render a template from within a view. This function accepts the request object, a template name, and a context dictionary that contains data that should be supplied to the template. This is just one illustration:

```
from django.shortcuts import render

def blog_post(request, post_id):
    post = get_blog_post_by_id(post_id)
    return render(request, 'blog/post_detail.html', {'post': post})
```

In this demonstration, the render function is applied to the 'blog/post_detail.html' template to render it with the context data. This data includes the fetched blog post. The HTML content will be constructed based on these data by the template afterward.

- Enhancing User Experience: Developers can craft web apps that operate faultlessly and provide users with a fantastic experience if they combine views and templates skillfully and thoughtfully. Utilizing Django's views and templates enables developers to design user interfaces that are captivating and engaging for site visitors. Customization, interactivity, and responsiveness are all within reach when using Django's views and templates.

THE DJANGO ADMIN PANEL

The administration of content within a web application is one of the most critical responsibilities, and the Django web development framework provides an instrument that is quite useful for accomplishing this goal: the Django Admin Panel. This component is packed with features that simplify content management by enabling developers and administrators to interact with the application's data effortlessly, streamlining the process of making updates and carrying out administrative activities. This post will investigate how the Django Admin Panel improves the development workflow and makes content administration more efficient (Schauble et al., 2017).

- **Built-In Administration Interface:** The Django Admin Panel is an administration interface ready to use out of the box and may be customized. It offers a straightforward method for users to interact with the data that the application stores. Developers can access and manage database records with a bit of setup, eliminating the need to write bespoke administrative views or interfaces.

- **Automatic CRUD Operations:** The Admin Panel can automate standard CRUD activities, which stands for create, read, update, and delete. It shows the data in tabular style, produces forms for creating and changing entries, and makes it easy for administrators to search, filter, and sort records. The amount of time spent on development is cut down by this automation, and maintenance is made more accessible.

- **Customization:** Even though it has a predefined user interface, the Django Admin Panel is adjustable. The developers can fine-tune the appearance and functionality to meet the application's requirements. Among the available customization possibilities are the specification of which database models are viewable, the creation of individualized views and forms, and the use of CSS styles to personalize the appearance and behavior.

- **Security and Access Control:** User authentication, role-based access control, and permission management are some of the sophisticated security features the Django Admin Panel enforces. Administrators can control who

can access the administrative interface and the actions that those users can carry out. This guarantees that sensitive data and essential operations will continue to be safeguarded.

- **Integration with Models:** The Admin Panel integrates without glitches or complications with the Django models. The models used to represent application data are defined by developers, and the Admin Panel is responsible for automatically generating an interface for managing those models. The management of content is made more efficient as a result of this tight integration.
- **Extensibility:** The Admin Panel's functionality can be expanded by developers by adding custom administrative views, filters, and actions. Because of its extensibility, the application can include specialized administrative tools and workflows explicitly designed to meet the requirements of its specific use case.
- **Multi-Language Support:** The Django Admin Panel supports multiple languages, making it suitable for use with applications aimed at users worldwide. Because it allows administrators to engage with material in their preferred language, it is an inclusive approach for managing applications that use internationalized language pairs.
- **Version Control and History:** The Admin Panel is responsible for keeping a record of the modifications made to the records, providing a history of the alterations, and offering the option to revert to earlier versions. This tool is quite helpful when it comes to auditing and being accountable.
- **Reduced Learning Curve:** The fact that the Django Admin Panel has a relatively shallow learning curve is one of its most significant selling points. It is optional to provide users with substantial training to quickly become skilled in content management if they already have a fundamental understanding of database records and online forms.
- **Time and Cost Efficiency:** Using the Django Admin Panel, development teams can drastically reduce the time and money often spent creating bespoke interfaces for managing content. It is no longer necessary for developers to spend time on mundane administrative responsibilities thanks to this technology, which frees them up to concentrate on the fundamental capabilities of the program.

User Authentication and Authorization

The authentication and authorization of users are essential components of the creation of online applications. Their purpose is to guarantee that only the appropriate users can access the proper resources while preserving data privacy and security (Thaduri, 2018). Django, a well-known framework for building websites, gives users access to a reliable and adaptable system for managing these functions.

User Authentication in Django: Django's authentication system makes verifying the users' identities much easier. The following are essential features:

- **User Models:** Django has a built-in user model that may be modified or expanded to include extra user-specific information. This model can also be used out of the box without additional configuration.
- **Registration and Login:** The framework offers a variety of user registration and login views and forms. Developers can include these in their respective applications.
- **Password Management:** Django manages passwords safely and securely, including hashing, resetting, and recovering forgotten passwords. It makes sure that passwords are not saved in plaintext, which is an improvement to the security of the system.
- **Session Management:** This feature ensures that user sessions are managed in an open and accessible manner, enabling user monitoring and consistent state maintenance throughout the various application pages.
- **Authentication Backends:** Django enables developers to design custom authentication backends to support various authentication methods. These methods include LDAP and OAuth.

User Authorization in Django: The process of declaring which actions or resources a user can access in Django is known as authorization. This is determined by the user's identification and their assigned roles. The following are essential features:

- **User Roles:** This enables role-based permissions to be assigned to users. Users can be put into groups with very particular permissions, allowing for highly granular control over who has access to what.
- **Decorators:** Using decorators, developers can restrict access to views based on user roles or conditions. For instance, the '@login_required' decorator guarantees that a view is only accessible to successfully authenticated users.
- **Object-level Permissions:** Django specifies rights on an object-by-object basis using "object-level permissions." This enables granular control over the individuals authorized to perform particular operations on particular items.
- **Middleware**: Middleware can be used to apply global access control rules, ensuring that certain users are diverted to different sections of the program or are prevented from accessing particular parts of the application.

Customization and Extensibility: It is possible to modify and extend Django, which is one of the strengths of this framework. Authentication and authorization logic explicitly tailored to an application can be created by its developers and implemented in that application. Because of this versatility, it is possible to develop sophisticated procedures for controlling access (Pippi, 2015).

FRONT-END INTEGRATION

In web development, integrating the front end, the user interface, with the backend logic is one of the most critical aspects. Django, a full-stack web framework, performs exceptionally well because it offers various tools and methods to smoothly connect front-end components with backend functionality. The integration of the front end in Django operates as follows:

- **Templates:** A templating engine is utilized by Django, which maintains a partition between the HTML code and the Python logic. Developers can create templates that contain placeholders for dynamic data, and then the data in these templates can be fetched from views and inserted into the template. Because of this separation, the code is easier to maintain, and front-end developers can concentrate solely on the presentation layer without worrying about the workings of the backend code (Yen & Yen, 2015).
- **Context Data:** In Django, the view's context data is transmitted to the template. This data, which may be variables, lists, or objects, renders dynamic content within the HTML template (Lal, 2016). This allows the displaying of database records, form inputs, and any other data required for the user interface.
- **Static Files:** A system for managing static files, including pictures, CSS, and JavaScript, is provided by Django. These files are kept in the static directory, conveniently accessible to connect to any needed HTML templates (Lal & Ballamudi, 2017). This ensures that the front end of your website can include styles and scripts that are effectively managed and served promptly.
- **Template Inheritance:** Django allows developers to design a base template that specifies a complete website's standard structure and layout. This strategy helps maintain coherence throughout the design process and reduces redundant coding (Lal et al., 2018).
- **Front-End Frameworks:** Integration with well-known front-end frameworks and libraries such as React, Vue.js, or Angular is possible with the Django web framework. Developers can construct single-page applications (SPAs) or interactive front-ends that use REST APIs to interface with the Django backend.
- **AJAX Integration:** AJAX, which stands for Asynchronous JavaScript and XML, is widely supported by Django, allowing for dynamic and asynchronous front-end interactions. It is possible to make asynchronous queries to Django views using JavaScript. These requests can return JSON or other data for the front end to process without requiring a complete website reload (Lal, 2019).
- **Front-End Libraries:** Django's modularity makes it possible to integrate front-end libraries and technologies developed by third parties, such as jQuery, Bootstrap, or D3.js. Thanks to these libraries, the user interface may be improved by adding pre-built components and interactive elements.
- **Front-End Testing:** Django's testing framework contains tools for testing the application's front end. Developers can write tests to ensure the application's user interface behaves as expected. This provides both the application's quality and its dependability.

SCALING AND PERFORMANCE OPTIMIZATION

As online applications expand their user bases and increase traffic, the need to scale those apps and optimize their performance becomes more pressing. Django is a robust framework that offers scalability and speed optimization capabilities, making it an ideal choice for addressing these difficulties. The following is a list of essential strategies and techniques for scaling Django and optimizing its performance:

- **Caching:** The mechanism of caching is a powerful tool for improving performance. Caching is a feature integrated into Django, allowing you to store data accessed frequently in memory or disk. As a result, there is less of a need to generate the same material often or query the database, leading to faster response times (Dauzon et al., 2016).
- **Database Optimization:** It is essential to perform database queries as efficiently as possible. Django's Object-Relational Mapping (ORM) component provides capabilities for improving the performance of database queries. The performance of a database can be considerably enhanced by the application of strategies.
- **Load Balancing:** Through load balancing, incoming traffic is distributed across numerous servers, preventing one server from becoming a bottleneck. Your application can be scaled horizontally to support more users if you install a load balancer in front of the Django instances it runs on.
- **Caching Reverse Proxies:** By caching and serving static content, reverse proxies that caches, such as Nginx or Varnish, can reduce the amount of work that your Django application needs to do. The response times for frequently requested content can be further improved thanks to the caching capabilities of these proxies, which can store responses from Django.

- **Asynchronous Processing:** Asynchronous views and tasks are supported by Django, which is an advantage when it comes to managing time-consuming procedures like the uploading of files or sending of emails. Processing data in an asynchronous manner releases server resources, which enables the application to manage a more significant number of requests at the same time (Thaduri, 2019).
- **Content Delivery Networks (CDNs):** CDNs store and serve static assets such as photos, CSS, and JavaScript from servers physically placed closer to the users. This decreases the amount of delay and speeds up the transmission of content. By integrating a content delivery network (CDN) with Django, you can ensure that your website pages will load more quickly.
- **Profiling and Optimization:** Performance bottlenecks can be found using profiling tools like the Django Debug Toolbar or Silk. You can identify sections of your application that could be improved by performing a profiling analysis, such as database queries or views that are too slow.
- **Gunicorn and uWSGI:** The application servers Gunicorn and uWSGI are trendy choices for deploying Django in production since they can manage a high volume of requests all at once effectively. These servers are typically utilized with a web server such as Nginx in a business setting (Lal, 2015).
- **Content Compression:** When content compression is enabled, such as with Gzip or Brotli, the data sent between the server and the client takes up less space. Payload sizes that are less significant lead to quicker load times for websites.
- **Horizontal Scaling:** Your Django application can have its capacity horizontally scaled to manage increased loads if you add extra servers or containers. Managing and deploying numerous instances of your application is much simpler because of technological advancements such as Docker and Kubernetes (George, 2016).

CONCLUSION

The Django web development framework is a vital cornerstone supporting the current-day internet. Because of its pragmatic design, precise code, and an abundance of built-in functionality, it has become the platform of choice for developers who are working to create web applications that are dynamic, safe, and efficient. Django substantially simplifies the process of both development and maintenance by adhering to the "Don't Repeat Yourself" (DRY) principle and implementing the Model-View-Controller (MVC) architectural pattern. This helps to reduce instances of repetition and improve the organization of the code. The Django Object-Relational Mapping (ORM) framework streamlines database interactions by eliminating the requirement for executing intricate SQL queries and fostering a more intuitive data management approach. It provides several sophisticated security features, including an authentication system and best practices for scalability, allowing web applications to expand without disruptions to keep up with changing requirements. Django's extensive library of reusable components, collectively called "apps," makes development more efficient by supplying ready-made solutions for typical web application functionalities. In addition, the framework offers various deployment choices, allowing it to be utilized in a wide range of hosting situations. Because of its adaptability and power, Django has emerged as a critical component in developing cutting-edge, feature-packed, and time-saving online applications in an era where web applications are indispensable for various business sectors and uses. Django has demonstrated time and again that it is more than capable of powering the ever-evolving environment of the internet, regardless of the size of the project or the complexity of the system being powered by it. Its rich history bodes well for its prospects in web development, which remain just as promising as those prospects.

REFERENCES

Cao, K., Wang, F., Liu, J. G. (2013). Study and Implementation of PM2.5 Data Download Service Based on Python. *Applied Mechanics and Materials*, *411-414*. https://doi.org/10.4028/www.scientific.net/AMM.411-414.555

Chen, S., Thaduri, U. R., & Ballamudi, V. K. R. (2019). Front-End Development in React: An Overview. *Engineering International*, *7*(2), 117–126. https://doi.org/10.18034/ei.v7i2.662

Dauzon, S., Ravindran, A., Bendoraitis, A. (2016). *Django: Web Development With Python.* Packt Publishing, Limited. Birmingham, GB.

Dekkati, S., & Thaduri, U. R. (2017). Innovative Method for the Prediction of Software Defects Based on Class Imbalance Datasets. *Technology & Management Review*, *2*, 1–5. https://upright.pub/index.php/tmr/article/view/78

Dekkati, S., Lal, K., & Desamsetti, H. (2019). React Native for Android: Cross-Platform Mobile Application Development. *Global Disclosure of Economics and Business*, *8*(2), 153-164. https://doi.org/10.18034/gdeb.v8i2.696

Dekkati, S., Thaduri, U. R., & Lal, K. (2016). Business Value of Digitization: Curse or Blessing?. *Global Disclosure of Economics and Business*, *5*(2), 133-138. https://doi.org/10.18034/gdeb.v5i2.702

Deming, C., Dekkati, S., & Desamsetti, H. (2018). Exploratory Data Analysis and Visualization for Business Analytics. *Asian Journal of Applied Science and Engineering*, *7*(1), 93–100. https://doi.org/10.18034/ajase.v7i1.53

Desamsetti, H., & Lal, K. (2019). Being a Realistic Master: Creating Props and Environments Design for AAA Games. *Asian Journal of Humanity, Art and Literature*, *6*(2), 193-202. https://doi.org/10.18034/ajhal.v6i2.701

George, N. (2016). *Mastering Django: Core.* Packt Publishing, Limited.

Hillar, G. C. (2016). *Building RESTful Python Web Services: Create Web Services That Are Lightweight, Maintainable, Scalable, and Secure Using the Best Tools and Techniques Designed for Python.* Packt Publishing, Limited. Birmingham, GB.

Hosen, M. S., Ahmmed, S., & Dekkati, S. (2019). Mastering 3D Modeling in Blender: From Novice to Pro. *ABC Research Alert*, *7*(3), 169–180. https://doi.org/10.18034/ra.v7i3.654

Lal, K. (2015). How Does Cloud Infrastructure Work?. *Asia Pacific Journal of Energy and Environment*, *2*(2), 61-64. https://doi.org/10.18034/apjee.v2i2.697

Lal, K. (2016). Impact of Multi-Cloud Infrastructure on Business Organizations to Use Cloud Platforms to Fulfill Their Cloud Needs. *American Journal of Trade and Policy*, *3*(3), 121–126. https://doi.org/10.18034/ajtp.v3i3.663

Lal, K. (2019). How Multiplayer Mobile Games have Grown and Changed Over Time?. *Asian Journal of Applied Science and Engineering*, *8*(1), 61–72. https://doi.org/10.18034/ajase.v8i1.56

Lal, K., & Ballamudi, V. K. R. (2017). Unlock Data's Full Potential with Segment: A Cloud Data Integration Approach. *Technology & Management Review*, *2*(1), 6–12. https://upright.pub/index.php/tmr/article/view/80

Lal, K., Ballamudi, V. K. R., & Thaduri, U. R. (2018). Exploiting the Potential of Artificial Intelligence in Decision Support Systems. *ABC Journal of Advanced Research*, *7*(2), 131-138. https://doi.org/10.18034/abcjar.v7i2.695

Lopez, C. F., Muhlich, J. L., Bachman, J. A., Sorger, P. K. (2013). Programming Biological Models in Python Using PySB. *Molecular Systems Biology*, *9*, 646. https://doi.org/10.1038/msb.2013.1

Pippi, M. (2015). *Python for Google App Engine: Master the Full Range of Development Features Provided by Google App Engine to Build and Run Scalable Web Applications in Python*. Packt Publishing, Limited. Birmingham, GB.

Ravindran, A. (2015). *Django Design Patterns and Best Practices: Easily Build Maintainable Websites with Powerful and Relevant Django Design Patterns.* Packt Publishing, Limited. Birmingham, GB

Schauble, S., Anne-Kristin, S., Bockwoldt, M., Pal, P., Heiland, I. (2017). SBMLmod: a Python-Based Web Application and Web Service for Efficient Data Integration and Model Simulation. *BMC Bioinformatics*, *18*. https://doi.org/10.1186/s12859-017-1722-9

Thaduri, U. R. (2017). Business Security Threat Overview Using IT and Business Intelligence. *Global Disclosure of Economics and Business*, *6*(2), 123-132. https://doi.org/10.18034/gdeb.v6i2.703

Thaduri, U. R. (2018). Business Insights of Artificial Intelligence and the Future of Humans. *American Journal of Trade and Policy*, *5*(3), 143–150. https://doi.org/10.18034/ajtp.v5i3.669

Thaduri, U. R. (2019). Android & iOS Health Apps for Track Physical Activity and Healthcare. *Malaysian Journal of Medical and Biological Research*, *6*(2), 151-156. https://mjmbr.my/index.php/mjmbr/article/view/678

Thaduri, U. R., Ballamudi, V. K. R., Dekkati, S., & Mandapuram, M. (2016). Making the Cloud Adoption Decisions: Gaining Advantages from Taking an Integrated Approach. *International Journal of Reciprocal Symmetry and Theoretical Physics*, *3*, 11–16. https://upright.pub/index.php/ijrstp/article/view/77

Yen, C. M., Yen, J. (2015). Cloud-Based Mechanical Design Oriented Python Program Development System. *Applied Mechanics and Materials*, *764-765*, 848-852. https://doi.org/10.4028/www.scientific.net/AMM.764-765.848

Zambelli, P., Gebbert, S., Ciolli, M. (2013). Pygrass: An Object Oriented Python Application Programming Interface (API) for Geographic Resources Analysis Support System (GRASS) Geographic Information System (GIS). *ISPRS International Journal of Geo-Information*, *2*(1), 201-219. https://doi.org/10.3390/ijgi2010201

--0--

Archive Link:

https://abc.us.org/ojs/index.php/ajtp/issue/archive