Bytexl’s guided project

Final Project report

|  |  |
| --- | --- |
| Name of the educator | Rudra Joshi |
| Project title | Timesheet Tracking Application |
| Tools / platforms used | Python, Django |

About the project: The Time-Entry Application is a project designed to help students understand how to build a functional time-tracking system. Students will implement features like user authentication, time logging with start and end times, timer functionality, and data analysis on time logs. This project mimics the needs of HR analytics tools, providing exposure to task logging, project management, and productivity insights.

**Software Requirements**

**1. Development Environment**

* **Python**: Version 3.8 or higher
* **Django**: Version 4.0 or higher for backend development
* **Django Rest Framework** (optional): For building APIs if necessary
* **Database**:
  + **SQLite**: For local development and testing
  + **PostgreSQL**: Recommended for production environments due to better scalability and reliability
* **HTML, CSS, and JavaScript**: For building the front-end interface
* **Bootstrap or Tailwind CSS** (optional): For responsive and mobile-friendly UI design

**2. Development Tools**

* **Code Editor**:
  + Recommended: VS Code, PyCharm, or Atom
* **Version Control**:
  + **Git**: For source code versioning and collaboration
  + **GitHub or GitLab**: For remote repository hosting and collaboration
* **Browser**:
  + **Google Chrome, Firefox, or Safari** for testing and debugging
* **Testing Tools** (optional):
  + **Postman**: For API testing if the application uses APIs
  + **Selenium or Django Test Framework**: For functional and integration testing

**3. Server Requirements**

* **Web Server**:
  + **Gunicorn**: WSGI HTTP server for running Django applications in production
  + **Nginx**: As a reverse proxy server for better handling of web traffic and load balancing
* **Deployment Platform**:
  + **Nimbus**: As specified for hosting the application
  + **Docker** (optional): For containerizing the application and ensuring consistent deployment environments

**4. Additional Libraries/Packages**

* **Django Extensions**: To support additional features like data export (e.g., CSV)
* **Celery with Redis or RabbitMQ** (optional): For handling asynchronous tasks like sending email notifications
* **Pillow**: For image processing if any image-related features are added to the project

**Hardware Requirements**

**1. Development Machine**

* **Processor**: Intel i5 or AMD equivalent (quad-core) or higher
* **RAM**: 8 GB minimum, 16 GB recommended for smoother operation when using multiple development tools and browser tabs
* **Storage**: At least 20 GB of free disk space for storing project files, databases, and additional software packages
* **Operating System**: Windows 10 or later, macOS, or any Linux distribution (Ubuntu, Fedora, etc.)

**2. Production Server (for Deployment)**

* **Processor**: Intel Xeon or AMD equivalent (dual-core or higher)
* **RAM**: 2 GB minimum for small-scale deployment, 4 GB or higher for handling larger traffic or multiple concurrent users
* **Storage**: 20 GB SSD minimum; higher capacity recommended for production to store application data and logs
* **Network**: Stable internet connection with adequate bandwidth, ideally on a VPS (Virtual Private Server) or cloud provider for reliability and scalability

**3. Optional Hardware (for Scalability)**

* **Load Balancer**: If handling heavy traffic, a load balancer like AWS ELB or Nginx can distribute traffic efficiently across servers.
* **Backup Storage**: External or cloud-based storage solution for regular backups to ensure data availability and integrity.

Functional requirements:

**1. User Authentication and Authorization**

* **User Registration**: Users should be able to create an account with a unique username and secure password.
* **Login/Logout**: Users must be able to log in and log out securely.
* **Role-Based Access Control**:
  + Regular users can log time entries and view their own data.
  + Admin users have additional privileges, such as viewing and managing all user time entries and accessing data analytics.

**2. Time Entry Management**

* **Add Time Entry**:
  + Users can log time spent on specific tasks with fields for project selection, task name, start time, and end time.
  + An option to use a timer that automatically records start and end times for tasks.
* **Edit and Delete Time Entries**: Users should have the ability to edit or delete their own time entries as needed.
* **View Time Entries**:
  + Users can view all their past entries in a list or table format.
  + Admin users can view and manage time entries for all users.

**3. Project and Task Management**

* **Project Selection**: Users should be able to select from a predefined list of projects when logging time.
* **Task Categorization**: Users should specify the task being worked on within a project.
* **Create and Edit Projects (Admin Only)**: Admin users should have the ability to add, edit, or delete projects and set their availability for user selection.

**4. Dashboard and Analytics**

* **Dashboard Overview**:
  + Users should have access to a dashboard displaying recent time entries and cumulative time logged.
* **Time Summary by Project**: Users should be able to view a summary of time spent per project, allowing for filtering by date (daily, weekly, monthly).
* **Data Visualization**: The dashboard should feature charts or graphs illustrating time distribution across projects for better productivity insights.

**5. Data Storage and Export**

* **Database Storage**: All time entries, projects, and user data should be stored securely in a database.
* **Data Export**: Users should have the ability to export their time entries as a CSV file for further analysis or record-keeping.

**6. Search and Filtering**

* **Filter by Date Range**: Users should be able to filter time entries by a specific date range for easier navigation.
* **Search by Project or Task**: Users should be able to search for specific entries by project name or task type.

**7. Notifications and Reminders (Optional)**

* **Daily or Weekly Reminders**: Option to send users reminders to log their time entries.
* **Email Notifications**: Notify users when certain events occur, such as reaching a project’s allocated hours or the submission of new time entries.

**8. User Profile Management**

* **Profile Information**: Users should be able to view and edit their profile information, including password updates.
* **Account Deletion**: Users should have an option to delete their account, which would remove their associated time entries.

**9. Application Settings (Admin Only)**

* **Manage App Settings**: Admin users should be able to configure application-wide settings, such as enabling/disabling features (e.g., reminders or data export).

**10. Deployment and Performance Optimization**

* **Fast Loading**: Optimize loading times for the application by using caching and optimizing assets.
* **SEO-Friendly Structure**: Ensure URL structure and metadata are optimized for search engine visibility.
* **Mobile Responsiveness**: The application should be accessible and usable on various devices, including desktops, tablets, and smartphones.

User interface requirements if any:NA

Inputs and Outputs:

**Inputs and Outputs**

**1. User Authentication and Authorization**

* **Input**:
  + User enters **username** and **password** for login or registration.
  + Admin or user logs out of the system.
* **Output**:
  + Successful login or registration leads to redirection to the dashboard or home page.
  + Error messages for invalid credentials, missing fields, or registration issues (e.g., username already taken).
  + Logout returns the user to the login page and clears their session.

**2. Time Entry Management**

* **Input**:
  + User submits a new time entry with details such as **project selection**, **task name**, **start time**, and **end time**.
  + User uses the timer to automatically record the **start** and **end times** for a task.
  + User edits or deletes an existing time entry.
* **Output**:
  + A success message confirms that the time entry has been added, edited, or deleted.
  + Updated list of time entries displayed, showing the latest changes in real-time.
  + Error messages for invalid time entries, such as end time being earlier than start time.

**3. Project and Task Management**

* **Input**:
  + Admin adds a new **project** or updates/deletes an existing project in the system.
* **Output**:
  + Updated list of projects available to users when logging time entries.
  + Confirmation messages for successful project additions, updates, or deletions.
  + Error messages if required fields are missing or if there are conflicts (e.g., trying to delete a project with active time entries).

List of subsystems:

 **User Authentication Subsystem**

* **Function**: Manages user registration, login, logout, and role-based access control.
* **Components**:
  + Registration Module: Allows new users to create accounts.
  + Login/Logout Module: Manages secure login sessions.
  + Role Management: Differentiates access levels for users and administrators.

 **Time Entry Management Subsystem**

* **Function**: Enables users to log, edit, and delete time entries for tasks.
* **Components**:
  + Time Entry Form: Interface for adding task name, project, start time, and end time.
  + Timer Function: Optional feature to track time automatically.
  + CRUD Operations: Supports creating, updating, and deleting time entries.

 **Project and Task Management Subsystem**

* **Function**: Provides administrators with tools to manage the list of projects and categorize tasks.
* **Components**:
  + Project Catalog: Admin-managed list of available projects.
  + Task Categorization: Allows users to classify tasks under specific projects.
  + Project CRUD Operations: Supports adding, editing, and removing projects.

 **Dashboard and Analytics Subsystem**

* **Function**: Displays user activity and time summaries in a visually organized way.
* **Components**:
  + Dashboard Interface: Main screen displaying recent time entries and summaries.
  + Data Visualization: Graphs and charts illustrating time distribution across projects.
  + Time Analysis Filters: Options to filter data by date range or project.

 **Data Storage and Export Subsystem**

* **Function**: Manages storage of all time entries, projects, and user information, with an option to export data.
* **Components**:
  + Database Management: Stores all entries in a structured, queryable format.
  + Export Module: Allows users to export time entries in CSV format for reporting or backup.

 **Search and Filtering Subsystem**

* **Function**: Facilitates searching and filtering of time entries to improve data access.
* **Components**:
  + Search Function: Enables keyword searches by project or task name.
  + Date Filtering: Allows filtering time entries by specific date ranges.
  + Filter by Project: Provides an option to view entries by individual project.

 **Notifications and Reminders Subsystem (Optional)**

* **Function**: Sends reminders and notifications to users to log time entries and view updates.
* **Components**:
  + Reminder Notifications: Optional email or in-app notifications prompting users to log time.
  + Alert Configurations: Settings to enable/disable notifications based on user or admin preferences.

 **User Profile Management Subsystem**

* **Function**: Allows users to manage and update their profile information.
* **Components**:
  + Profile Editor: Interface for users to update personal information and change passwords.
  + Account Management: Provides options for users to delete their accounts if needed.

 **Admin Settings and Configuration Subsystem**

* **Function**: Provides application-wide configurations and settings for administrators.
* **Components**:
  + Application Settings: Allows admins to enable or disable features like reminders and exports.
  + Role Management: Manages permissions and access levels for all users.
  + Application Configuration: Configures settings related to data privacy and security.

 **Deployment and Optimization Subsystem**

* **Function**: Ensures smooth deployment, performance optimization, and maintenance.
* **Components**:
  + Deployment Configuration: Configures application settings for deployment on platforms like Nimbus.
  + Performance Tuning: Manages caching, asset optimization, and SEO setup for fast loading.
  + Security Management: Ensures that the application remains secure and complies with best practices.

Other Applications relevant to your project: HRMS, Inventory Management System, HIMS.

Designing of Test cases:

**TITLE STEPS**

|  |  |
| --- | --- |
| User Login | 1. Open login page 2. Enter valid credentials 3. Click login button |
| Admin Dashboard Access | 1. Login as admin 2. Navigate to dashboard 3. Verify access to admin functions |
| User Registration | 1. Open registration page 2. Enter required information 3. Submit registration form |
| Scroll Up Button | 1. Scroll down the webpage 2. Locate and click the 'Scroll Up' button |
| Admin Page Access | 1. Login as an admin 2. Navigate to the admin page 3. Verify access permissions |
| Contact Us Form Submission | 1. Open Contact Us page 2. Fill out the form with name, email, and message 3. Submit the form |
| Team Page and Gallery Display | 1. Navigate to the Team page 2. Verify the display of team members 3. Open the gallery section |

Future Work: Integration with HRMS and ATS(Applicant Tracking Systems).

References: NA

Reflection of the project creation:

**1. Technical Challenges Encountered**

During the development of the Timesheet-Tracking Application, several technical challenges were encountered:

* **User Authentication and Security**: Implementing secure user authentication and role-based access control was challenging. Ensuring that only authorized users could access or modify data required careful handling of user sessions, password encryption, and validation.
* **Accurate Time Tracking**: Designing the timer feature to record start and end times accurately was complex, particularly when handling cases where users could leave or refresh the page. It required persistence of time data and syncing across the session to ensure consistency.
* **Data Visualization**: Creating meaningful and real-time data visualizations for time entries was another challenge, especially in generating charts that accurately display data from different date ranges and projects. Selecting and integrating a suitable charting library into Django without disrupting page load times was critical.
* **Data Validation and Error Handling**: Ensuring data consistency in time entries (e.g., end time must be after start time) required implementing validations at both the front-end and back-end levels. Additionally, handling and displaying errors to guide users in correcting their input was essential for user experience.
* **Deployment and Optimization**: Deploying on a live server and optimizing for performance presented its own challenges, including configuring settings for secure sessions, caching data, and enabling SEO features for better visibility.

**2. Addressing Challenges with Existing Software Engineering Knowledge**

Several software engineering techniques and prior knowledge helped address these challenges:

* **Modularization and Code Reusability**: By organizing the code into modular components (views, models, and templates), it was easier to handle each functionality independently, making it manageable to debug and extend individual components.
* **Use of Design Patterns**: Familiarity with MVC (Model-View-Controller) architecture in Django streamlined the separation of concerns, which was particularly helpful when handling user authentication and database interactions separately from the UI and business logic.
* **Error Handling and Validation**: Prior experience with input validation and error handling allowed us to enforce strict checks at both client-side (JavaScript) and server-side (Django forms). This ensured data integrity, particularly for the time-entry logs.
* **Version Control**: Using Git for version control helped manage code changes efficiently, especially when resolving issues or making improvements iteratively. Branching in Git allowed testing new features independently without impacting the main codebase.
* **Deployment Best Practices**: Familiarity with deployment processes, like setting up environments and configuring caching and load balancing, helped address the complexities of deploying a production-ready Django application.