**🧩 Problem Statement:**

**"Design a Productivity Enhancer for an Individual working from 9:00 AM to 5:00 PM."**

**📝 Brief Description:**

In today's fast-paced corporate environment, particularly for IT Engineers, juggling multiple responsibilities within a standard 9:00 AM to 5:00 PM workday can be overwhelming. The core challenge revolves around effectively managing and prioritizing:

1. **Email Inbox Overload:** Constantly checking and responding to a deluge of emails, which often distracts from core tasks.
2. **Meeting Marathon:** Attending numerous scheduled and ad-hoc meetings, which consume significant time and break workflow.
3. **Jira Task Management:** Ensuring timely completion of assigned tasks tracked in project management tools like Jira, often requiring focused, uninterrupted work.

The problem is to design and develop a solution that helps an individual navigate these demands, minimize context-switching, maintain focus, and ultimately enhance their overall productivity without extending work hours. The goal is to provide a tool that intelligently assists them in allocating their time, prioritizing urgent items, and ensuring all critical tasks (emails, meetings, Jira tickets) are addressed within the stipulated workday.

**🧾 Use Case Objective:**

Build a smart **dashboard assistant** for an individual that intelligently organizes their 9:00 AM – 5:00 PM work schedule by combining data from email, calendar, and JIRA.

**📥 Input:**

1. **Email Feed**:
   * Access to user’s email inbox (e.g., Gmail API or Microsoft Graph API)
   * Categorized by meetings, action items, and unread emails
2. **Calendar Data**:
   * Meeting schedules (start time, end time, participants)
   * Free time blocks detection
3. **JIRA Tasks**:
   * User's assigned tasks (via JIRA REST API)
   * Task priority, status, deadlines, and estimated time
4. **User Preferences**:
   * Focus time preferences (e.g., best time for deep work)
   * Notification settings
   * Priority settings for different types of emails/tasks.
   * Daily/weekly goals.

**📤 Output:**

1. **Smart Daily Planner**:
   * Suggested task schedule with recommended time slots
   * Alerts for upcoming meetings & deadlines
2. **Integrated Dashboard**:
   * To-do list pulled from JIRA
   * Quick links to join meetings
   * Email summary section (unread, important)
3. **Productivity Report**:
   * Hours spent in meetings vs. focused task time
   * Task progress (e.g., 70% completed JIRA issues)
4. **Focus Suggestions**:
   * Suggest 25-min Pomodoro blocks during free slots
   * Mute distractions during deep work periods

**📊 Sample Dashboard Design:**

**🌐 Dashboard Sections:**

| **Section** | **Description** |
| --- | --- |
| **Today’s Timeline** | Visual 9–5 schedule with color-coded blocks: Emails, Meetings, Tasks |
| **JIRA To-do List** | Task list pulled from JIRA with status, ETA, and start button |
| **Email Summary** | Smart filtered view (Unread, Urgent, Actionable) |
| **Meetings** | Calendar view of all meetings with links to join |
| **Focus Zone** | Suggested focus sessions with countdown timer and "Do Not Disturb" mode |
| **Analytics** | Daily/weekly report (time in meetings, tasks completed, focus time, etc.) |

**🔧 Suggested Tech Stack:**

**💻 Backend:**

* **Java, Node.js** or **Python Flask/FastAPI**
* Authentication (OAuth for Gmail, Microsoft, JIRA)

**📡 APIs to Integrate:**

* **Gmail / Outlook Email API**: for reading emails
* **Google Calendar / Microsoft Graph**: for meetings
* **JIRA REST API**: for tasks
* **OpenAI API (optional)**: to summarize long emails or meeting transcripts

**📱 Frontend:**

* **Angular or React.js** with **Tailwind CSS** for a modern dashboard UI
* **Chart.js / Recharts** for productivity analytics
* **FullCalendar.js** for calendar display

**🛠 Dev Tools:**

* GitHub for version control
* Postman for testing APIs
* Docker for containerizing app
* Firebase / MongoDB / PostgreSQL for storing user preferences and state

**🔍 Evaluation Criteria for Hackathon:**

| **Criteria** | **Weightage** |
| --- | --- |
| Problem Understanding | 10% |
| UI/UX Design | 20% |
| API Integration | 20% |
| Task Scheduling Logic | 20% |
| Innovation & Features | 20% |
| Presentation & Demo | 10% |

**Use Case: Work Allocation for Project Task Management**

This document outlines a new hackathon use case designed to challenge IT graduates to develop innovative solutions for optimizing work allocation within project teams.

**Problem Statement: "Work Allocation to Manage the Project Task"**

In a typical IT project, efficient task allocation is crucial for timely delivery and optimal resource utilization. However, project managers often face significant challenges in assigning tasks effectively. The core problem statement for this hackathon is to create a solution that intelligently manages project task allocation by considering:

1. **Resource Availability:** Accurately assessing each team member's current workload, planned leaves, and capacity to take on new tasks for the duration of the project or sprint.
2. **Skills & Ability Matching:** Identifying and matching task requirements with individual team members' skills, expertise, and historical performance (ability). This includes considering primary and secondary skills.
3. **Task Dependencies & Priorities:** Understanding the interdependencies between tasks and their respective priorities to ensure critical paths are not blocked and high-priority items are addressed by the most suitable resources.

The goal is to develop a tool that streamlines the work allocation process, reduces manual effort for project managers, prevents resource over-allocation or under-utilization, and ultimately enhances project efficiency and team productivity.

**Explanation of the Problem**

Consider a scenario where a project manager has a backlog of tasks in a tool like Jira. Traditionally, they might manually assign tasks based on a superficial understanding of who is "free" or who "usually does this type of work." This often leads to:

* **Uneven Workload Distribution:** Some team members are overloaded, leading to burnout and delays, while others are under-utilized.
* **Skill Mismatches:** Tasks requiring specialized skills are assigned to less experienced individuals, impacting quality and increasing rework.
* **Missed Deadlines:** Critical tasks are delayed because the assigned resource is already committed or lacks the necessary skills, impacting project timelines.
* **Lack of Transparency:** It's difficult to get a real-time overview of who is working on what, their progress, and their remaining capacity.

A smart work allocation system would address these issues by providing a data-driven approach to assignment, ensuring the right person is working on the right task at the right time.

**Input and Output**

**Input**

The solution should consume detailed information about projects, tasks, and resources. Key inputs include:

* **Resource Profiles:**
  + Employee ID, Name (e.g., "Rohit Sharma").
  + Primary Skills (e.g., "React.js", "Node.js", "Java", "SQL", "DevOps").
  + Secondary Skills (e.g., "Python", "Cloud - AWS", "Agile Coaching").
  + Experience Level (e.g., "Junior", "Mid", "Senior", "Lead").
  + Current Availability (e.g., daily hours available for new work, upcoming leave dates).
  + Current Workload (e.g., tasks assigned, estimated effort remaining on current tasks).
  + Historical Performance Data (e.g., average task completion time for specific task types, bug rates - *optional but highly valuable*).
* **Project Task Data (from Jira or similar PM tool):**
  + Task ID, Title, Description.
  + Estimated Effort (e.g., in hours or story points).
  + Due Date.
  + Priority (e.g., "Highest", "High", "Medium", "Low").
  + Required Skills (e.g., "React.js", "API Development").
  + Dependencies (e.g., "Task A must complete before Task B starts").
  + Task Type (e.g., "Feature", "Bug", "Enhancement", "Documentation").
* **Project/Sprint Configuration:**
  + Project Start/End Dates.
  + Sprint Start/End Dates.
  + Team Size and composition.
  + Overall project goals and key milestones.
* **Manager Overrides/Preferences:**
  + Manual adjustments for specific assignments.
  + Rules for skill development (e.g., assign a junior engineer to a task with a senior mentor).

**Output**

The primary output of the solution should be an optimized work allocation plan and actionable insights presented through an intuitive dashboard. This includes:

* **Optimized Task Assignments:**
  + A list of recommended task-to-resource assignments for a given period (e.g., daily, weekly, or sprint).
  + Justification for assignments (e.g., "Assigned to Priya Singh due to high React.js skill and 70% availability").
* **Resource Utilization Reports:**
  + Visualizations showing each team member's projected workload and availability over time (e.g., "Rohit Sharma: 90% utilized this week").
  + Identification of over-allocated or under-utilized resources.
* **Skill Gap Analysis:**
  + Identification of tasks that cannot be assigned due to a lack of available skills within the team.
  + Suggestions for upskilling or hiring.
* **Dependency Management View:**
  + A visual representation of task dependencies and their status, highlighting potential bottlenecks.
* **What-If Scenarios:**
  + Ability to simulate different allocation strategies and observe their impact on timelines and resource utilization.
* **Allocation History & Performance:**
  + Tracking of past allocations and their success rates (e.g., tasks completed on time by assigned resource).

**Dashboard Design**

The work allocation dashboard should provide project managers with a comprehensive, real-time view and control over task assignments.

1. **Team Overview (Top Panel):**
   * **Resource Cards:** Each card represents a team member (e.g., "Priya Singh").
     + Display: Name, Primary Role, Current Utilization Percentage (e.g., a progress bar).
     + Status Indicator: Available, Busy, On Leave.
     + Quick View: Number of tasks assigned, next 2-3 tasks.
   * **Overall Project Progress:** A high-level view of sprint or project completion percentage.
2. **Task Queue / Unassigned Tasks (Left Sidebar):**
   * A scrollable list of unassigned tasks, perhaps categorized by priority or due date.
   * Each task item shows: Task ID, Title, Estimated Effort, Required Skills, Due Date.
   * Drag-and-drop functionality to assign tasks to resources in the Team Overview or Allocation Grid.
3. **Allocation Grid / Timeline View (Main Central Area):**
   * A gantt-chart-like or calendar-like grid showing resource names on one axis and time (days/weeks) on the other.
   * Tasks are visually placed on the timeline under the assigned resource, indicating their duration.
   * Color-coding for task priority or status.
   * Visual alerts for over-allocation (e.g., a red highlight on a resource's timeline if their capacity is exceeded).
   * Ability to click on a task to view details or reassign.
4. **Skills Matrix & Gap Analysis (Right Sidebar/Collapsible Panel):**
   * A table or graph showing the team's collective skills and areas where certain skills are lacking for upcoming tasks.
   * Highlights tasks that cannot be assigned due to skill unavailability.
5. **Notifications & Alerts:**
   * A small notification area for alerts like "Rohit Sharma is 120% utilized this week," "Task #JIRA-123 is unassigned and due tomorrow," or "Skill gap detected for 'Machine Learning' in upcoming sprint."
6. **Filter & Search:**
   * Ability to filter tasks by priority, skill, or type.
   * Search functionality for tasks and resources.
7. **Reports & Analytics (Dedicated Section):**
   * **Utilization Heatmap:** A visual representation of team member utilization over a longer period.
   * **Skill Concentration Chart:** Showing distribution of skills across the team.
   * **"What-If" Scenario Simulator:** A dedicated interface to test different assignments and see their immediate impact on timelines and utilization.

**Possible Technologies**

Hackathon participants can leverage a wide range of technologies to build their solutions.

**Frontend (User Interface)**

* **Web-based (Highly Recommended):**
  + **HTML, CSS, JavaScript:** Core web technologies.
  + **Frameworks/Libraries:**
    - **React.js:** For building dynamic and interactive user interfaces with reusable components (e.g., for resource cards, task items).
    - **Vue.js:** Another progressive framework known for its simplicity.
    - **Angular:** A comprehensive framework suitable for complex enterprise applications.
  + **Styling Frameworks:**
    - **Tailwind CSS:** For rapid UI development with utility-first classes, ensuring responsiveness.
    - **Chakra UI / Material-UI (for React):** Component libraries for pre-built, accessible UI elements.
  + **Charting Libraries:**
    - **Chart.js / Recharts (for React):** For visualizing utilization, skill distribution, and progress.
  + **Gantt Chart Libraries:**
    - **DHTMLX Gantt / React-Gantt-Timeline:** For creating the interactive allocation grid/timeline.

**Backend (Logic & Data Processing)**

* **Programming Languages:**
  + **Python:** Excellent for complex logic, data processing, optimization algorithms (e.g., for allocation), and integration with machine learning.
  + **Node.js (JavaScript):** Ideal for building scalable RESTful APIs and handling real-time updates.
  + **Java (Spring Boot):** Robust for large-scale, mission-critical applications.
* **Frameworks:**
  + **Flask / Django (Python):** For building the backend APIs and processing allocation logic.
  + **Express.js (Node.js):** For handling API requests and responses.

**Database**

* **NoSQL Databases:**
  + **Firebase Firestore:** Real-time, serverless NoSQL database, excellent for quickly storing and synchronizing resource profiles, tasks, and allocation data.
  + **MongoDB:** Flexible document database for storing complex resource and task structures.
* **SQL Databases:**
  + **PostgreSQL:** Robust and feature-rich relational database, good for structured data and complex queries if relationships are critical.

**APIs & Integrations**

* **Project Management APIs:**
  + **Jira REST API:** Crucial for fetching tasks, updating statuses, and potentially assigning tasks programmatically.
  + **Asana API / Trello API:** If participants choose to integrate with other PM tools.
* **Calendar/HRIS APIs (Optional but enhances availability tracking):**
  + **Google Calendar API / Microsoft Graph API:** For fetching team members' actual availability or planned leaves.
* **Authentication & Authorization:**
  + **Firebase Authentication:** For managing user logins (project managers).
  + **OAuth 2.0:** For securely integrating with external services like Jira.
* **AI/ML (Highly Recommended for Advanced Solutions):**
  + **Constraint Programming/Optimization Libraries (Python):** Such as PuLP, OR-Tools, or custom algorithms to solve the task allocation problem as an optimization problem (balancing skills, availability, and priorities).
  + **Machine Learning (e.g., Scikit-learn, TensorFlow/PyTorch):**
    - For predicting task completion times based on historical data.
    - For recommending the "best fit" resource for a task based on skill similarity and past success.
    - For identifying potential bottlenecks or overloads proactively.
  + **Natural Language Processing (NLP):**
    - To extract keywords or skill requirements from task descriptions if not explicitly tagged.

**Deployment**

* **Cloud Platforms:**
  + **Firebase Hosting / Google Cloud Platform (GCP):** For deploying the web frontend and potentially serverless functions (Cloud Functions for backend logic).
  + **Vercel / Netlify:** For fast and easy deployment of frontend applications.
  + **AWS (EC2, Lambda) / Microsoft Azure (App Service, Azure Functions):** For more complex backend deployments.

**Test Cases with Sample Data**

Here are some test cases with sample data, using Indian names, to illustrate how the system should function.

**Scenario 1: Basic Skill Match & Availability**

* **Resources:**
  + **Priya Singh:** Skills: React.js (Senior), Node.js (Mid). Availability: 40 hrs/week. Current Workload: 20 hrs.
  + **Amit Kumar:** Skills: Node.js (Senior), SQL (Mid). Availability: 40 hrs/week. Current Workload: 10 hrs.
* **Tasks:**
  + **T1: Build User Profile UI (High Priority):** Est. Effort: 15 hrs. Required Skill: React.js. Due: End of week.
  + **T2: Develop Backend API (Medium Priority):** Est. Effort: 20 hrs. Required Skill: Node.js. Due: End of week.
* **Expected Outcome:**
  + T1 assigned to Priya Singh (Senior React skill, sufficient availability: 40 - 20 = 20 hrs free > 15 hrs needed).
  + T2 assigned to Amit Kumar (Senior Node.js skill, sufficient availability: 40 - 10 = 30 hrs free > 20 hrs needed).
  + Both resources are appropriately utilized.

**Scenario 2: Over-allocation Detection**

* **Resources:**
  + **Deepak Sharma:** Skills: Java (Senior), Spring Boot (Senior). Availability: 40 hrs/week. Current Workload: 35 hrs.
  + **Anjali Devi:** Skills: Java (Mid), SQL (Mid). Availability: 40 hrs/week. Current Workload: 10 hrs.
* **Tasks:**
  + **T3: Refactor Auth Module (High Priority):** Est. Effort: 20 hrs. Required Skill: Java. Due: End of week.
* **Expected Outcome:**
  + System should flag that Deepak Sharma is the best skill match but assigning T3 would over-allocate him (35 + 20 = 55 hrs).
  + System might suggest Anjali Devi, even if she's Mid-level, or recommend delaying T3, or splitting it if possible. A warning about potential over-allocation should be prominent.

**Scenario 3: Dependency Handling & Skill Development**

* **Resources:**
  + **Vikram Patel:** Skills: Python (Senior), ML (Senior). Availability: 40 hrs/week. Current Workload: 5 hrs.
  + **Neha Gupta:** Skills: Python (Junior). Availability: 40 hrs/week. Current Workload: 0 hrs.
* **Tasks:**
  + **T4: Data Preprocessing (Medium Priority):** Est. Effort: 10 hrs. Required Skill: Python. Due: Mid-next week. (No dependencies)
  + **T5: Model Training (High Priority):** Est. Effort: 25 hrs. Required Skill: Python, ML. Depends on: T4. Due: End of next week.
* **Expected Outcome:**
  + T4 is assigned first. System might suggest Neha Gupta for skill development (if manager preference allows), with Vikram Patel as a mentor. Or, if speed is critical, assign T4 to Vikram.
  + T5 *must* be assigned to Vikram Patel due to ML skill requirement and T4 dependency. It should only be allocated after T4's completion is projected.

**Scenario 4: Leave/Unavailability**

* **Resources:**
  + **Kavita Rao:** Skills: React.js (Senior). Availability: 40 hrs/week. Current Workload: 0 hrs. Upcoming Leave: Next Monday to Wednesday (3 days).
* **Tasks:**
  + **T6: Implement Dashboard Widget (High Priority):** Est. Effort: 20 hrs. Required Skill: React.js. Due: End of next week.
* **Expected Outcome:**
  + System should recognize Kavita's reduced availability next week.
  + If T6 is assigned, the system should account for her reduced working days (e.g., 2/5 days available) and adjust the estimated completion time accordingly, or suggest assigning to another available React resource if the deadline is tight.

These test cases cover various aspects of resource management, helping participants validate their allocation logic.