Spider R&D

Software Task 2

Greetings from SPIDER R&D!

Welcome to the **SPIDER R&D** Inductions ‘25! Over the upcoming weeks, you will take on a series of tasks designed to help you learn and explore different areas of technology and research.

The inductions will be conducted in two stages:

1. Task 1
2. Task 2

Getting Started

The tasks are designed to cater to a range of skill levels, ensuring that even beginners can learn and progress. While you may encounter challenges along the way, perseverance is key. It is normal to face difficulties; overcoming them will be part of your learning experience.

Support and Resources

If you run into any issues or need assistance, do not hesitate to reach out to any member of the club. We are here to help you navigate through the tasks and ensure a fruitful learning journey.

General Instructions:

● Carefully read and comprehend the problem statement before you begin.Misunderstandings and incorrect completion of tasks will not be acceptable. Do not hesitate to seek clarification if you're unsure about any aspect.
● The goal is for you to understand the underlying concepts and write your own code. While it's okay to look up code snippets for reference, directly copying someone else's code is forbidden.
● For complex tasks, divide them into smaller, manageable modules. Focus on completing each module individually. This approach makes the task easier and ensures steady progress and a sense of accomplishment as you complete each part.
● The Internet is a powerful tool. Use it to search for solutions to your queries and doubts. Most questions can be resolved with a quick search. However,make sure you understand the solutions you find and can explain them.
● If you encounter any issues or have any questions, feel free to contact us.

You can post your queries in the WhatsApp group or send a private message to one of us. We are here to support you and provide guidance as needed.
● Write clear and concise documentation for your code. Use comments to explain the purpose of complex code sections. This will not only help others understand your work but also aid you in recalling your thought process later.
● Test your code thoroughly to ensure it works as expected. Debug any issues that arise and verify that all parts of your code function correctly. Consider edge cases and test your code under different scenarios.
● Use version control systems like Git to keep track of changes in your code. This allows you to revert to previous versions if something goes wrong.

- The deadline for Task 2 is **2nd July 2025**.

We are excited to have you on board and look forward to see your progress. Good luck from the Spider R&D family! 🙂

Application Development

Bill Split

Note: You can develop either a web application or a mobile application for completing the task

Sanjai loves going out on outings and spontaneous tours with his friends. But there's just one thing that turns his fun adventures into a nightmare - splitting the bill. From figuring out who paid for the food to tracking ticket purchases, Sanjai ends up stressed at the end of every trip. Overwhelmed, he dreams of an application that can handle all the bill splitting and settling with ease. Help Sanjai's dreams come true by coding the application (with a working frontend and backend) with the below mentioned features.

Demo Video Link: **Spider Application Dev Demo - Task 2**

Level 1

- Implement authentication with secure login and logout functionality using JWT or session-based authentication.
- Implement a feature where a user can search for other usernames and friend/unfriend another user.
- Users must be able to create a bill splitting group and be able to add only their friends to the group. The group creator must be able to remove people from the group also.
- Members of the group must be able to add expenses with a category (as shown in the demo video), and the app should split the bill equally among all members. Show how much the user owes to other members of the group and how much others owe them.
- The user who added an expense in a group must have an option to delete the expense also.
- Allow the group creator to delete the group.

Level 2

- Allow users to view, and update their profile page.
- Enable users to upload and update their profile picture.
- Implement password reset/recovery.
- Display pie charts and circular charts, inside every group, for category-wise expenses, as shown in the demo video.

- Implement a feature that allows users to send a summary of all expenses in a group, including category and amount details, to their email address.
- Add a custom splash screen. Also, check if the user is connected to the internet, if not then display an error screen.

Level 3

- Implement OAuth login using passport.js (eg. Google OAuth, Facebook OAuth, etc).
- Implement push notifications in the application. Notifications should come when a new expense is added in a group and when a friend request comes in.
- Implement a feature that allows users to split expenses unequally among selected group members. When adding an expense, provide options to split the amount either by **percentage** (e.g., A pays 60%, B 30%, C 10%) or by **exact amounts** (e.g., A pays ₹100, B pays ₹50, C pays ₹150)
- Implement a feature where users can choose which group members are involved in a specific expense. Even if a group contains five members, the user should be able to select only a subset (e.g., 3 out of 5) for the expense. Use a **checkbox UI** or similar interface to allow users to select participants during the expense creation process, enabling accurate and relevant bill splitting.

Note:

- Completing all features of Level 1 will only be considered as completion of the task.
- Completing Level 2 and Level 3 features will give you an edge.
- You may use third-party libraries and third-party APIs for implementation.
- If you are making a web application, make it mobile responsive.
- If you are using React Native, use CLI and not Expo.

Tasks will be evaluated based on:

- Completion level of task
- UI/UX
- Responsiveness
- Code simplicity
- Creativity

Resources:

Authentication

JWT Authentication with Node.js:
https://www.digitalocean.com/community/tutorials/api-authentication-with-json-web-tokensjwt-and-passport
Passport.js Documentation (JWT, Session, OAuth): https://www.passportjs.org/
OAuth 2.0 Fundamentals:
https://auth0.com/docs/get-started/authentication-and-authorization

Frontend (Web)

React Official Documentation: https://react.dev/learn
FreeCodeCamp React Tutorial:
https://www.freecodecamp.org/learn/front-end-development-libraries/#react
React Hooks Guide: https://www.robinwieruch.de/react-hooks/

Frontend (app)

React Native: Style · React Native

Flutter: UI | Flutter

Backend (Node.js & Express)

Node.js Official Guide: https://nodejs.org/en/docs/guides/
Express.js Documentation: https://expressjs.com/
MongoDB with Node.js:
https://www.mongodb.com/developer/languages/javascript/node-js-mongodb-user-management/
REST API Design with Express:
https://www.freecodecamp.org/news/rest-api-design-best-practices-build-a-rest-api/

Database (MongoDB)

MongoDB Official Tutorials: https://www.mongodb.com/docs/manual/tutorial/
Mongoose ODM Guide: https://mongoosejs.com/docs/
MongoDB CRUD Operations:
https://www.geeksforgeeks.org/mongodb-crud-operations/

Push Notifications

Web Push API Tutorial:
https://www.smashingmagazine.com/2017/02/push-notifications-guide/

Responsive Design

Tailwind CSS Documentation: https://tailwindcss.com/
Responsive Web Design Basics:
https://developer.mozilla.org/en-US/docs/Web/Development/Mobile/Responsive_design

DevOps

You have already containerized your full-stack application comprising a React frontend, Rust backend, and PostgreSQL database using Docker and Docker Compose. The next step is to prepare your application for production deployment with a focus on security, scalability, and automation.

This task will challenge your understanding of reverse proxies, Docker image optimization, horizontal scalability, and CI/CD automation — all essential for running modern cloud-native applications in production.

Level 1: Infrastructure Configuration

1. Reverse Proxy with Nginx and HTTPS

Set up an Nginx container that:

- Serves the React frontend at the root path (/)
- Proxies API requests (/api) to the Rust backend
- Enforces HTTPS using self-signed SSL certificates
- Handles HTTP-to-HTTPS redirection
- Manages CORS headers properly
- Enables gzip compression for performance
- Add the nginx service in docker-compose.

Level 2: Automation via CI/CD

2. Jenkins CI/CD Pipeline

Automate your development workflow with Jenkins:

- Run linters and tests for both the frontend and backend on every push and pull request
- Build and push Docker images for all services to Docker Hub on every push to the main branch
- Automatically deploy updated services using Docker Compose upon successful image builds

Level 3: Production Optimization

This level encourages you to optimize your application and follow production-ready best practices, including:

- Docker Image & Container Optimization

- Nginx Performance Tuning
- CI/CD Pipeline Enhancements
- Security Best Practices in Docker, Nginx, Jenkins as well
- Observability & Resiliency in Docker and Nginx

Submission Guidelines

- Use the same GitHub repository from Task 1 to push all updated code and configuration files
- Add a well-structured README.md explaining the setup, architecture, and workflow with screenshots and screen records as well.
- Share the GitHub repository link as your final submission
- Level 1 and Level 2 must be fully implemented for the task to be considered completion.
- Thoughtful and creative solutions in Level 3, especially those following industry best practices, will receive special recognition.

Resources:

1. Jenkins:

- https://www.youtube.com/watch?v=6YZvp2GwT0A
- https://medium.com/@prateek.malhotra004/essential-jenkins-best-practices-for-developers-streamline-your-ci-cd-process-dc3aa38f9928
- https://www.jenkins.io/doc/book/pipeline/pipeline-best-practices/

2. Nginx:

- https://nginx.org/en/docs/
- https://www.youtube.com/watch?v=iInUBOVeBCc
- https://medium.com/@mathur.danduprolu/securing-your-web-server-with-nginx-https-and-best-practices-part-5-7-99ad19bf5b1f

3. Docker

- https://docs.docker.com
- https://snyk.io/blog/10-docker-image-security-best-practices/
- https://sysdig.com/learn-cloud-native/dockerfile-best-practices/
- https://docs.docker.com/compose

Blockchain Development

Zero-Knowledge Voting System for Student Council Elections

Every year, students elect the **President of the Student Council**, a role that represents the entire student body. Currently, the voting process is closed and controlled by a central admin: ballots are sent over email, voting takes place in a physical lab, and students never see how many votes each candidate actually received — they only hear who won.

This raises concerns around transparency and lack of verifiability. To address this, the student body wants a better way — a **blockchain-based voting system** that ensures:

- **Privacy of each voter's choice**
- **Transparency of results**
- **Fairness without centralized trust**

Your task is to build a full-stack zero-knowledge-powered decentralized voting system for electing the Student Council President.

Level 1

- Authenticate users using **wallet login** (e.g., MetaMask).
- Create a simulated **whitelist of eligible voters** (wallets pre-approved to vote).
- Display a list of candidates for the **Student Council President** election.
- Each student can cast **only one vote** per election.
- Show a **public and verifiable vote count** for each candidate after voting ends, while keeping voter identities private.
- Build a clean **frontend dashboard** for viewing candidates, voting, and seeing live results.

Level 2

- Implement voting using a **zero-knowledge protocol** to **Prevent double voting and Preserve anonymity** of who voted for whom
- Enable **admin (e.g., election officer)** to: Start and end elections, Upload eligible voter list, Add/remove candidates
- Allow students to **generate a zero-knowledge proof** that they're eligible (e.g., “I’m a student”) without revealing their identity.

Level 3

- Support **ranked-choice voting** (vote for top 3 preferences) with ZK-proof tallying.
- Provide **voting receipts** (e.g., a hash of their anonymous vote) that voters can use to prove they voted — without showing what they voted for. Or enable issuance of a **"Voted" NFT badge** (optional Soulbound) after successful voting, without revealing who was voted for.
- Store candidate manifesto or campaign video on **IPFS**, linked to their public profile.

Note:

- Completing all features of Level 1 and some features of Level 2 will only be considered as completion of the task.
- Completing Level 2 and Level 3 features will give you an edge.

Tasks will be evaluated based on:

- Completion level of task
- Smart Contract logic
- Smart Contract Security
- Code simplicity
- Creativity

Resources:

1. https://updraft.cyfrin.io/courses/fundamentals-of-zero-knowledge-proofs
2. https://www.rareskills.io/zk-book

Networking:

Firewall System

Real-Time Monitoring and Domain/Port-Based Filtering on a Local System

In this task, you will **simulate** a basic firewall and traffic monitoring system using your own machine. You'll develop a **Python-based system** that captures and inspects packets, logs meaningful traffic data, and enforces access control policies.

Everything in **Levels 1,2** will be done on a **single system (your machine)**.For bonus task use a VM .

Level 1: Packet Sniffing (Logging Traffic)

Goal: Passively monitor and log network activity on your system.

- Use Python with `pyshark` to sniff live traffic.
- Log:
    - Source and destination IP addresses
    - Source and destination ports
    - HTTP Host headers (for HTTP traffic)

Output Required : Pcap Files .

Level 2:Real-Time Access Control (Filtering & Blocking)

Goal: Enforce basic access control by filtering traffic based on content.

Enhance your system to:

- **Detect**:

    - HTTP requests to vulnerable http websites .
    - Traffic to sensitive ports (e.g., `23` for Telnet, `21` for FTP)

- **Block**:
    - Detected packets in real-time using either:
        - `netfilterqueue` + Python script
        - `iptables` rules with helper scripts

- **Log**:
    - Every blocked request attempt with reason (e.g., "Blocked domain", "Blocked port")

Output Required : Python script along with log files .

Bonus Task: Website Hosting & Multi-VM Integration

Set up a **separate VM** for hosting websites using Apache.

On the Hosting VM:

- Create 7 name-based virtual hosts(i.e host 7 websites from your local vm):
    - `site1.local`, `site2.local`, ..., `site7.local`
- Each serves unique HTML content
- Configure `/etc/hosts` so your system can resolve those `.local` names

On your main system:

- Connect to the Hosting VM via network
- Test:

- `site1.local` to `site5.local` — should load successfully
- `site6.local` and `site7.local` — should be blocked by your firewall

Output Required : Screenshots of the two vmware setup and pcap files which show data transfer between the two vms .

Evaluation Criteria

- Real-time sniffing and meaningful logging
- Accurate filtering logic for domains and ports
- Organized logs and modular Python code
- Successful test results and integration with hosted websites (Bonus)

Resources:

https://www.digitalocean.com/community/tutorials/how-to-set-up-apache-virtual-hosts-on-ubuntu-20-04

https://www.geeksforgeeks.org/how-to-use-apache-webserver-to-host-a-website/

CyberSecurity:

Level 1: Basic Recon Automation

Objective: Automate fundamental recon tasks to replace manual steps. Make a Python tool that automatically finds basic information about a Domain.

Steps & Tasks:

1. **Domain Input**: Accept a domain via CLI argument or prompt.
2. **Subdomain Enumeration**: Use:
    - **crt.sh API** (API Docs),
    - **Sublist3r** (GitHub).
3. **DNS Record Lookup**: Retrieve A, NS, MX records using:
    - **dnspython** (Documentation),
    - **dnsrecon** (GitHub).
4. **WHOIS Information**: Fetch via:
    - **python-whois** (PyPI),
    - CLI fallback to `whois` command.
5. **HTTP Headers:** Fetch and display server banners using `requests` or `curl -I`.
6. **robots.txt & sitemap.xml:** Retrieve and display `/robots.txt` and `/sitemap.xml` content.
7. **GeoIP Lookup:** Determine server location using a free IP geolocation API.
8. **Output:** Display results on terminal or write to `basic_<domain>.txt`.

Deliverables:

- `basic_recon.py` script

- Sample output file (e.g., `example.com_basic.txt or screenshot of terminal with results`)
- `README.md` with setup instructions and usage guide

Level 2: Intermediate Recon Toolkit

Objective: Extend Level 1 by adding deeper scanning, intelligence gathering, and structured reporting.

Steps & Tasks:

1. **Modular Script:** Use Python with `argparse` to enable/disable modules individually.
2. **Port Scanning & Banner Grabbing:**
    - Perform port scanning using `nmap` (with `python-nmap`) or `masscan`.
    - Extract banners using `--script=banner` or socket connections.
3. **Technology Detection:** Identify web technologies using `WhatWeb` or Wappalyzer API.
4. **Email Harvesting:** Use `theHarvester` or implement Google/Bing scraping for email IDs.
5. **Shodan Lookup:** Query Shodan API for detailed open service info.
6. **Structured Report:** Export all gathered data in JSON or CSV format to `reports/<domain>.json` or `.csv`.

Deliverables:

- `intermediate_recon.py` script
- Sample structured report (`example.com_report.json` or `.csv`)
- Updated `README.md` with command-line arguments and example usage

Level 3: Advanced Recon Suite

Objective: Package the toolkit into a polished, professional-grade tool with a focus on reporting, UX, and automation. Bonus features include Web UI and Dockerization.

Steps & Tasks:

1. **Live Screenshots Module:**
    - Capture live screenshots of discovered subdomains using `gowitness` or `Selenium`.
    - Save to `reports/screenshots/<domain>_<timestamp>.png`.
2. **WAF/CDN Detection:**
    - Identify Web Application Firewalls and CDN services using `wafw00f` and HTTP header analysis.
    - Store findings in a dedicated security section in the final report.
3. **Vulnerability Scanning (Optional):**
    - Use tools like `Nikto` or `OpenVAS` for basic vulnerability scanning.
    - Summarize critical findings in `reports/vuln/summary.csv`.
4. **Report Generation:**
    - Export final recon summary as a well-designed HTML report using `Jinja2`, Markdown + `grip`, or static templates.
    - Include sections for each module with visual elements (charts, tables, collapsible panels).

Bonus :

Flask Web UI:

- Create a web dashboard using Flask:
    - Input domain and select modules through the interface
    - View live logs and status
    - Navigate structured recon results in browser
- Structure:
    - Templates in `templates/`
    - Static assets in `static/`
    - Flask routes in `app.py`

Dockerization:

- Dockerize the tool for seamless deployment and portability:
    - Add a `Dockerfile` to install all dependencies

Deliverables:

- `your_tool_name/` directory with all code and modules
- Final HTML report template and output samples
- `README.md` including:
    - Setup, dependencies, and installation
    - Description of all modules and their usage
    - Bonus section for Flask Web UI and Docker (optional)
- Screenshots or usage video (optional but encouraged)

UI UX

You can choose **any one Problem statement** and work on it. The UI part includes figma prototype.The UX part includes:

1. Problem statement analysis and key solutions proposed
2. User personas
3. User flow of the app
4. User journey

PS - 1 Academic Tracker Mobile Application

You are required to conceptualize and design a mobile application that helps students efficiently track deadlines, exam dates, and assignment submissions.
This app should centralize academic schedules and notifications to improve students' time management and academic performance.

Functional prototype using Figma/InVision
Include subtle animations/transitions
For calendar views, task completion, or notification modals.

Scope of Work:

User Research & Personas

Identify the target users (e.g., school/college students)
Briefly describe their goals, frustrations, and needs:

- Goals: Stay organized, meet deadlines, balance workload.
- Frustrations: Scattered information, last-minute reminders.
- Needs: Centralized, easy-to-access academic timeline on mobile.

Information Architecture

Define clear categories for navigation:

- By Course
- By Deadline
- By Priority

Include a userflow diagram seperately showing navigation flow:

- Home/Dashboard
- Calendar View
- Task/Assignment Details
- Notifications
- Settings

Wireframes

Low-fidelity wireframes for at least 3 key mobile screens:

- Home/Dashboard (Upcoming deadlines and overview)
- Calendar View (Deadlines, exams, assignments visualized)
- Task/Assignment Detail Page (Detailed view with status update option)

High-Fidelity UI Screens

Design at least 4 mobile screens with attention to layout, color, typography, and interactivity:

- Home/Dashboard
- Calendar View
- Task/Assignment Detail Page
- Notifications Page

Mobile Responsiveness

Design primarily for mobile screen sizes (Android/iOS)
Ensure touch-friendly UI, easy thumb navigation, and clear call-to-action buttons.

Style Guide

Include the chosen color palette, fonts, button styles, icons, spacing system, and navigation patterns.
Include subtle animations/transitions for:

- Adding or completing tasks
- Notification pop-ups
- Calendar interactions

Bonus:
Brownie points for a landing screen with progress tracking animations or an interactive,

PS-2 Self care app

Design a functional prototype of a self care mobile app for people who lead busy and stressful lives and forget to care for themselves which is critical for promoting their own physical , mental,emotional,and social health and well being.

User Research & Personas

- Identify typical users (e.g., young professionals, students, caregivers)
- Briefly describe their goals, stress points, daily routines, and self-care challenges

Information Architecture

- Define key app sections: e.g., Daily Check-In, Habit Tracker, Mindfulness Tools, Journaling, Community

- Include a **basic user flow** diagram (e.g., user opens app → logs mood → gets recommendation)

Wireframes

Create low-fidelity wireframes for **at least 3 core screens**:

- Home / Dashboard
- Self-Care Activity Page (e.g., breathing exercise, journaling prompt)
- Habit Tracker or Reminder Setup Page

High-Fidelity UI Screens

Design **at least 4 polished screens** that focus on layout, clarity, and emotional tone:

- Welcome / Onboarding Screen
- Home Dashboard (shows today's check-ins, reminders, tips)
- Activity Page (e.g., meditation, stretch, hydration log)
- Progress or Reflection Page (summary of user's habits, mood logs, or journal entries)

5. Mobile Responsiveness

- Design for **mobile view (mandatory)**
- **Tablet support** is optional but appreciated

6. Style Guide

- Provide a mini style guide showing:
  • Color palette (calm, uplifting tones)
  • Fonts used (readable, soft tone)
  • Button styles, iconography, input fields
  • Animation examples (e.g., smooth transitions, loading states, reward microinteractions)