CS50x HarvardX: Modelo de Plano de Projeto Final

Autor: Manus AI

Introdução

Este modelo de plano de projeto foi criado para ajudar a estruturar o seu Projeto Final do CS50x. Um plano bem definido é crucial para o sucesso de qualquer empreendimento de software, ajudando a organizar ideias, gerir o tempo e garantir que todos os requisitos sejam atendidos. Adapte este modelo à sua ideia específica de projeto.

1. Título do Projeto

Sistema de Gestão de Tarefas Inteligente (To-Do List Avançada) com CS50x

2. Descrição do Projeto

Este projeto consiste no desenvolvimento de uma aplicação web robusta para gestão de tarefas, que permitirá aos utilizadores criar, organizar, priorizar e acompanhar as suas atividades diárias de forma eficiente. O problema que resolve é a dificuldade em manter o controlo de múltiplas tarefas e prazos, oferecendo uma solução centralizada e intuitiva. Os utilizadores-alvo são indivíduos, estudantes e pequenos grupos que procuram uma ferramenta personalizável para aumentar a sua produtividade. O objetivo principal é criar uma plataforma que não só ajude na organização pessoal, mas que também possa ser estendida para funcionalidades de colaboração. O impacto esperado é melhorar a gestão do tempo e a conclusão de tarefas para os seus utilizadores, demonstrando a aplicação prática dos conceitos de ciência da computação na resolução de problemas do mundo real.

3. Tecnologias a Utilizar

As tecnologias principais serão Python para o backend, Flask como framework web, e HTML, CSS e JavaScript para o frontend. Para a base de dados, será utilizado SQLite, que é leve e adequado para este tipo de aplicação. O Git será usado para controlo de versão.

- **Linguagens:** Python, JavaScript, HTML, HTML, CSS
- **Frameworks/Bibliotecas:** Flask, Bootstrap (para o frontend, se desejado para agilizar o design responsivo), Jinja2 (motor de templates do Flask).
- **Base de Dados:** SQLite
- **Outras Ferramentas:** Git (para controlo de versão), VS Code (ambiente de desenvolvimento).

4. Funcionalidades Principais

O projeto terá as seguintes funcionalidades principais, focando num produto mínimo viável que pode ser expandido:

- **Autenticação de Utilizadores:** Registo, login e logout seguros para gerir contas de utilizadores.
- **Gestão de Tarefas:**
  - **Criação de Tarefas:** Adicionar novas tarefas com título e descrição.
  - **Visualização de Tarefas:** Listar todas as tarefas do utilizador.
  - **Edição de Tarefas:** Modificar o título e a descrição de tarefas existentes.
  - **Eliminação de Tarefas:** Remover tarefas.
- **Priorização de Tarefas:** Atribuir níveis de prioridade (e.g., Alta, Média, Baixa) às tarefas.
- **Prazos (Due Dates):** Definir datas de conclusão para as tarefas.
- **Marcação de Conclusão:** Marcar tarefas como concluídas/incompletas.
- **Filtragem e Ordenação Básica:** Filtrar tarefas por estado (concluídas/pendentes) e ordenar por prazo ou prioridade.

5. Estrutura de Dados e Algoritmos

A estrutura de dados central será uma base de dados relacional (SQLite), com tabelas para `Utilizadores` e `Tarefas`. As relações entre estas tabelas permitirão associar tarefas a utilizadores. Para a manipulação e recuperação de dados, serão utilizadas consultas

SQL. Algoritmos de ordenação e filtragem serão aplicados para apresentar as tarefas de forma organizada (por exemplo, ordenar por prazo, prioridade ou estado de conclusão).

- **Estruturas de Dados:** Tabelas SQL (para Utilizadores e Tarefas), Dicionários Python (para dados temporários e configurações).
- **Algoritmos:** Algoritmos de ordenação (para listar tarefas por prioridade, prazo), algoritmos de pesquisa (para encontrar tarefas específicas), hashing (para senhas de utilizadores).

6. Design e Arquitetura (se aplicável a aplicações web)

A arquitetura será cliente-servidor. O frontend será responsável pela interface de utilizador e interação, enquanto o backend Flask gerenciará a lógica de negócio, a interação com a base de dados e a autenticação. A comunicação entre frontend e backend será via requisições HTTP. O design será responsivo para garantir usabilidade em diferentes dispositivos.

- **Frontend:** HTML para a estrutura, CSS (potencialmente com Bootstrap para agilizar o design responsivo) para o estilo, e JavaScript para interatividade do lado do cliente (e.g., validação de formulários, atualizações dinâmicas).
- **Backend:** Flask para gerir as rotas, a lógica de negócio, a interação com a base de dados SQLite e a autenticação de utilizadores.
- **Comunicação:** Requisições HTTP (GET, POST) para troca de dados entre frontend e backend.

7. Plano de Desenvolvimento (Timeline Sugerida)

O desenvolvimento será dividido nas seguintes fases, com um prazo final a 31 de dezembro de 2025:

- **Fase 1: Configuração e Autenticação (1-2 semanas)**
    - Configurar o ambiente de desenvolvimento Flask e a base de dados SQLite.
    - Implementar o registo, login e logout de utilizadores.
    - Criar o modelo da base de dados para utilizadores e tarefas.
- **Fase 2: Gestão Básica de Tarefas (2-3 semanas)**
    - Desenvolver as funcionalidades CRUD (Criar, Ler, Atualizar, Eliminar) para tarefas.
    - Implementar a visualização da lista de tarefas para o utilizador autenticado.
    - Adicionar campos para título e descrição da tarefa.

- **Fase 3: Funcionalidades Avançadas de Tarefas (2-3 semanas)**
    - Implementar a priorização de tarefas (Alta, Média, Baixa).
    - Adicionar a funcionalidade de prazos (due dates).
    - Permitir marcar tarefas como concluídas.
    - Implementar filtragem e ordenação básica das tarefas.
- **Fase 4: Refinamento e Documentação (1-2 semanas)**
    - Melhorar a interface de utilizador (UI) e a experiência do utilizador (UX) com CSS/Bootstrap.
    - Realizar testes exaustivos e corrigir bugs.
    - Preparar o ficheiro `README.md` detalhado para o GitHub, explicando o projeto, como configurá-lo e usá-lo.
    - Revisão final do código e preparação para a entrega.

8. Desafios Potenciais e Soluções

- **Desafio 1: Gestão de Estado no Frontend:** Manter a interface de utilizador sincronizada com o backend pode ser complexo, especialmente com múltiplas interações do utilizador.
    - **Solução Potencial:** Utilizar JavaScript para fazer requisições assíncronas (AJAX/Fetch API) para atualizar partes da página sem recarregar, e gerir o estado da aplicação de forma cuidadosa no frontend.
- **Desafio 2: Segurança da Aplicação:** Garantir que a aplicação está segura contra vulnerabilidades comuns, como injeção de SQL e Cross-Site Scripting (XSS).
    - **Solução Potencial:** Utilizar prepared statements ou um ORM (como SQLAlchemy, se desejado) para evitar injeção de SQL. Escapar todas as entradas do utilizador antes de as renderizar no HTML para prevenir XSS. Seguir as melhores práticas de segurança do Flask.
- **Desafio 3: Design Responsivo:** Garantir que a aplicação funciona bem em diferentes tamanhos de ecrã (desktop, tablet, telemóvel).
    - **Solução Potencial:** Utilizar um framework CSS como o Bootstrap, que já vem com um sistema de grelha responsivo, ou escrever media queries em CSS para adaptar o layout a diferentes larguras de ecrã.

9. Honestidade Académica

Confirmo que este projeto respeitará integralmente a política de honestidade académica do CS50x. Todo o código será de autoria própria ou devidamente atribuído, e qualquer colaboração seguirá estritamente as diretrizes do curso para garantir a integridade do trabalho.

Este plano é um guia. Sinta-se à vontade para ajustá-lo conforme o seu projeto evolui. O mais importante é manter-se organizado e focado nos objetivos do seu projeto final.