Project Expectation Document: AI-Powered Conversational LMS

Project Title: AI-Powered Conversational Learning Management System (LMS)

Objective:
Build a web-based Learning Management System that allows users to create accounts, log in, and engage in conversational learning with an AI tutor. The AI tutor will ask users what they want to learn, process their input, and provide tailored explanations, examples, and follow-up questions to enhance understanding.
Project Scope:

- Develop a Django-based backend for user authentication, session management, and progress tracking.
- Create a React-based frontend for user interaction (account creation, input forms, and conversational interface).
- Integrate an AI model (via API or open-source LLM) to handle conversational learning.
- Implement progress tracking to store user learning history and quiz results.
- Deploy the application on Vercel for accessibility and scalability.
- Use Git for version control and collaboration.

Target Audience:
Students, lifelong learners, or anyone seeking a conversational learning experience for topics like programming, history, or science.
Deliverables:

- A fully functional web application with user authentication and a conversational AI tutor.
- A database to store user accounts, learning progress, and quiz results.
- A responsive React frontend for seamless user interaction.
- Integration with an AI model (e.g., xAI's Grok API or Hugging Face LLM) for conversational learning.
- Source code hosted on GitHub with proper documentation (README, setup instructions).
- Deployed application on Vercel with a live URL.
- Project report summarizing development process, challenges, and outcomes (optional, for portfolio).

Success Criteria:

- Users can create accounts, log in, and interact with the AI tutor.
- The AI tutor responds accurately to user inputs (e.g., "I want to learn Python") with clear explanations and follow-up questions.
- User progress (topics covered, quiz scores) is saved and retrievable.
- The application is deployed on Vercel, accessible via a public URL, and performs reliably.
- Code is well-documented, modular, and pushed to GitHub with a clear commit history.

Technologies:

- Backend: Django (Python) for user management and database integration.
- Frontend: React (JavaScript) for user interface.
- AI: xAI Grok API (or Hugging Face Transformers for open-source LLM).

- Database: PostgreSQL (or SQLite for development).
- Version Control: Git and GitHub.
- Hosting: Vercel for deployment.
- Other Tools: Python libraries (e.g., requests for API calls, nltk or spacy for NLP), Node.js for frontend tooling.

Constraints:

- Must use open-source tools or free-tier APIs where possible to minimize costs.
- AI responses should be safe, accurate, and appropriate for educational use.
- Deployment on Vercel should leverage their free tier for hosting.
- Project must be completed within 10 weeks.

Assumptions:

- The mentee has access to a computer with Python, Node.js, and Git installed.
- Free-tier APIs (e.g., xAI Grok, Hugging Face) provide sufficient usage for development.
- Public datasets or APIs (e.g., Wikipedia, OER) are available for educational content.
- The mentee is familiar with Git workflows and Vercel deployment.

Risks and Mitigation:

- Risk: Limited experience with LLMs or APIs.
  Mitigation: Use well-documented APIs (e.g., xAI Grok) and tutorials from Hugging Face.
- Risk: Overly complex AI responses or content sourcing.
  Mitigation: Start with simple topics (e.g., Python basics) and use pre-trained models or APIs.
- Risk: Deployment issues on Vercel.
  Mitigation: Follow Vercel's Django/React deployment guides and test locally first.

Stakeholders:

- Mentee: Responsible for development, testing, and deployment.
- Mentor (You): Provides guidance, reviews progress, and suggests improvements.
- End Users: Learners who will use the LMS for education.

Documentation Requirements:

- README.md in GitHub repository with setup instructions, dependencies, and usage guide.
- Code comments for complex logic (e.g., AI integration, quiz generation).
- Optional: A brief report or presentation summarizing the project for portfolio purposes.

Implementation Guide for Each Phase

Below is a detailed guide on how your mentee can implement each phase of the project, including open-source tools, platforms, hosting on Vercel, and Git workflows. Given their proficiency in Python, JavaScript, React, Node.js, Django, Laravel, and PHP, they have the skills to execute this project efficiently.

Phase 1: Project Setup, Django Backend, and User Authentication (Weeks 1-2)

Objective: Set up the project structure, initialize a Django backend, and implement user authentication.

Implementation Steps:

- Project Setup:
    - Create a new directory: mkdir ai-lms && cd ai-lms.
    - Initialize a Git repository: git init.
    - Create a .gitignore file to exclude venv/, node_modules/, and .env.
    - Set up a virtual environment: python -m venv venv && source venv/bin/activate (Linux/Mac) or venv\Scripts\activate (Windows).
    - Install Django: pip install django.
    - Create a requirements file: pip freeze > requirements.txt.
- Django Backend:
    - Start a Django project: django-admin startproject lms_backend.
    - Create a Django app for user management: python manage.py startapp users.
    - Configure settings.py to include the users app and set up PostgreSQL (or SQLite for simplicity):

```
python
DATABASES = {
  'default': {
    'ENGINE': 'django.db.backends.sqlite3',
    'NAME': 'mydatabase',
  }
}
```

-
    - Install django-allauth for authentication: pip install django-allauth.
    - Configure django-allauth in settings.py for email-based login/registration.
- User Authentication:
    - Create models in users/models.py for user profiles (e.g., name, email, learning preferences).
    - Use Django's built-in authentication or django-allauth for login, signup, and password reset.
    - Create views and URLs for login (/login), signup (/signup), and logout (/logout).
    - Test authentication locally: python manage.py runserver.

Tools/Platforms:

- Django: Open-source web framework for backend (https://www.djangoproject.com).
- django-allauth: Open-source package for authentication (https://github.com/pennersr/django-allauth).
- PostgreSQL: Free database (https://www.postgresql.org) or SQLite for development.
- Git: For version control (https://git-scm.com).

Git Workflow:

- Create a GitHub repository: Go to https://github.com, click "New," and name it ai-lms.
- Link local repo to GitHub: git remote add origin <github-repo-url>.
- Commit initial code: git add . && git commit -m "Initial Django setup with authentication".
- Push to GitHub: git push -u origin main.

Hosting (Not Yet): Hosting will be addressed in Phase 5.

Phase 2: AI Model Integration and Basic Conversational Flow (Weeks 3-4)
Objective: Integrate an AI model to process user inputs and deliver conversational learning responses.
Implementation Steps:

- Choose an AI Model:
    - Option 1: xAI Grok API (recommended for simplicity):
        - Sign up at https://x.ai/api for API access (free tier may be available).
        - Obtain an API key and store it in a .env file using python-dotenv: pip install python-dotenv.
        - Example API call using requests:

```
python
import requests
import os
from dotenv import load_dotenv

load_dotenv()
api_key = os.getenv("GROK_API_KEY")
response = requests.post(
  "https://api.x.ai/v1/grok",
  headers={"Authorization": f"Bearer {api_key}"},
  json={"input": "Explain Python functions"}
)
print(response.json())
```

    - Option 2: Hugging Face Transformers (open-source):
        - Install Transformers: pip install transformers.
        - Use a pre-trained model like distilbert-base-uncased or facebook/blenderbot-400M-distill:

```
python
from transformers import pipeline
chatbot = pipeline("conversational", model="facebook/blenderbot-400M-distill")
response = chatbot("I want to learn Python")
print(response)
```

- Basic Conversational Flow:
    - Create a Django view to handle user inputs (e.g., POST request with topic: “I want to learn Python”).
    - Pass the input to the AI model and return the response to the user.
    - Example view in users/views.py:

```
python
from django.http import JsonResponse
import requests
from django.views.decorators.csrf import csrf_exempt
import json

@csrf_exempt
def learn(request):
    if request.method == "POST":
        data = json.loads(request.body)
        topic = data.get("topic")
        # Call AI API (e.g., Grok)
        response = requests.post(
            "https://api.x.ai/v1/grok",
            headers={"Authorization": f"Bearer {api_key}"},
            json={"input": f"Explain {topic} in simple terms"}
        )
        return JsonResponse({"response": response.json()})
    return JsonResponse({"error": "Invalid request"}, status=400)
```

    - Add URL in urls.py: path('learn/', views.learn, name='learn').
- Content Sourcing:
    - Use Wikipedia API for supplemental content: pip install wikipedia-api.
    - Example:

python
```python
import wikipediaapi
wiki = wikipediaapi.Wikipedia('en')
page = wiki.page("Python (programming language)")
summary = page.summary[:500]  # Get first 500 chars
```

Tools/Platforms:

- xAI Grok API: For conversational AI (https://x.ai/api).
- Hugging Face Transformers: Open-source NLP models (https://huggingface.co).
- Wikipedia API: Free content source (https://github.com/martin-majlis/Wikipedia-API).
- python-dotenv: For secure API key management (https://github.com/theskumar/python-dotenv).

Git Workflow:

- Create a branch: git checkout -b ai-integration.
- Commit changes: git add . && git commit -m "Integrated AI model with basic conversational flow".
- Push branch: git push origin ai-integration.
- Create a pull request on GitHub and merge into main.

Phase 3: React Frontend and Conversational UI (Weeks 5-6)

Objective: Build a React frontend for user interaction, including account creation and conversational interface.

Implementation Steps:

- Set Up React:
    - Create a React app in a separate directory: npx create-react-app lms-frontend && cd lms-frontend.
    - Install dependencies: npm install axios (for API calls to Django).
    - Start the React app: npm start.
- Authentication UI:
    - Create components for login/signup (e.g., Login.js, Signup.js).
    - Use axios to call Django authentication endpoints:

javascript
```javascript
import axios from 'axios';
```

```
const login = async (email, password) => {
  const response = await axios.post('http://localhost:8000/login/', { email, password });
  return response.data;
};
```

- *Conversational UI:*
  - *Create a Learn.js component with an input field and chat-like interface.*
  - *Call the Django /learn/ endpoint to interact with the AI:*

```
javascript
import { useState } from 'react';
import axios from 'axios';

const Learn = () => {
  const [topic, setTopic] = useState('');
  const [response, setResponse] = useState('');

  const handleSubmit = async () => {
    const res = await axios.post('http://localhost:8000/learn/', { topic });
    setResponse(res.data.response);
  };

  return (
    <div>
      <input
        type="text"
        value={topic}
        onChange={(e) => setTopic(e.target.value)}
        placeholder="What do you want to learn?"
      />
      <button onClick={handleSubmit}>Learn</button>
      <p>{response}</p>
    </div>
  );
};
export default Learn;
```

- **Styling:**
  - Use CSS or a library like Tailwind CSS (npm install tailwindcss and follow setup: https://tailwindcss.com/docs/installation).

Tools/Platforms:

- React: Open-source frontend framework (https://reactjs.org).
- Axios: For API calls (https://github.com/axios/axios).
- Tailwind CSS: Optional for styling (https://tailwindcss.com).
- Node.js: For React tooling (https://nodejs.org).

Git Workflow:

- Create a branch: git checkout -b frontend.

- Commit changes: git add . && git commit -m "Added React frontend with conversational UI".
- Push branch: git push origin frontend.
- Merge via pull request.

Phase 4: Progress Tracking and Quiz Functionality (Weeks 7-8)

Objective: Add progress tracking and quiz features to store user learning history and reinforce learning.

Implementation Steps:

- Progress Tracking:
    - Create a model in users/models.py for progress:

```python
from django.db import models
from django.contrib.auth.models import User

class Progress(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    topic = models.CharField(max_length=200)
    content = models.TextField()
    quiz_score = models.IntegerField(default=0)
    timestamp = models.DateTimeField(auto_now_add=True)
```

    - Create views to save/retrieve progress:

```python
def save_progress(request):
    if request.method == "POST":
        data = json.loads(request.body)
        progress = Progress.objects.create(
            user=request.user,
            topic=data['topic'],
            content=data['content'],
            quiz_score=data.get('quiz_score', 0)
        )
        return JsonResponse({"status": "success"})
```

- Quiz Functionality:
    - Generate simple quizzes using the AI model (e.g., prompt Grok to create questions: “Generate 3 multiple-choice questions about Python functions”).
    - Store quiz results in the Progress model.
    - Update the React frontend to display quizzes and submit scores:

```javascript
const Quiz = ({ topic }) => {
  const [questions, setQuestions] = useState([]);
  const [score, setScore] = useState(0);

  const fetchQuiz = async () => {
    const res = await axios.post('http://localhost:8000/learn/', {
      topic: `Generate 3 multiple-choice questions about ${topic}`,
    });
    setQuestions(res.data.response);
  };

  return <div>{/* Render quiz questions */}</div>;
};
```
Tools/Platforms:

- Django ORM: For database operations.
- PostgreSQL: For storing progress data.

Git Workflow:

- Create a branch: git checkout -b progress-quiz.
- Commit changes: git add . && git commit -m "Added progress tracking and quiz functionality".
- Push branch: git push origin progress-quiz.
- Merge via pull request.

Phase 5: Testing, Debugging, Deployment on Vercel, and Documentation (Weeks 9-10)
Objective: Test the application, deploy it on Vercel, and finalize documentation.
Implementation Steps:

- Testing:
    - Test authentication: Ensure login/signup works.
    - Test AI responses: Verify the AI provides accurate, relevant answers.
    - Test progress tracking: Confirm data is saved/retrieved correctly.
    - Use Django's testing framework: python manage.py test.
    - Test React frontend with npm test.
- Deployment on Vercel:
    - Backend (Django):
        - Install gunicorn: pip install gunicorn.
        - Create a Procfile: web: gunicorn lms_backend.wsgi.
        - Update settings.py for production:

```
python
ALLOWED_HOSTS = ['*']
DEBUG = False
```

-
    -
        - Push backend to a separate GitHub repo: ai-lms-backend.
        - Deploy on Vercel:
            - Sign up at https://vercel.com.
            - Import the ai-lms-backend repo.
            - Set environment variables (e.g., GROK_API_KEY) in Vercel's dashboard.
            - Configure Vercel to use Python and Procfile.
    - Frontend (React):
        - Build the React app: npm run build.
        - Push frontend to a separate GitHub repo: ai-lms-frontend.
        - Deploy on Vercel:
            - Import the ai-lms-frontend repo.
            - Set the build command to npm run build and output directory to build.
            - Add environment variable for Django backend URL (e.g., REACT_APP_API_URL=https://ai-lms-backend.vercel.app).
    - Test the live URL (e.g., https://ai-lms-frontend.vercel.app).

- **Documentation**:
    - Create a README.md:

markdown
```
# AI-Powered Conversational LMS
A web-based LMS with a conversational AI tutor.

## Setup
1. Clone repo: `git clone <repo-url>`
2. Backend: `cd lms_backend`, `pip install -r requirements.txt`, `python manage.py migrate`
3. Frontend: `cd lms_frontend`, `npm install`, `npm start`
4. Set environment variables: `GROK_API_KEY`

## Deployment
Deployed on Vercel: <frontend-url>
```

Tools/Platforms:

- Vercel: Free hosting for frontend and backend (https://vercel.com).
- GitHub: For code hosting (https://github.com).
- Gunicorn: For Django production server (https://gunicorn.org).

Git Workflow:

- Create a branch: git checkout -b deployment.
- Commit changes: git add . && git commit -m "Final testing and deployment setup".
- Push branch: git push origin deployment.
- Merge into main and tag release: git tag v1.0.0 && git push origin v1.0.0.

Additional Notes

- Open-Source Resources:
    - Use free-tier APIs (xAI Grok, Hugging Face) to keep costs at zero.
    - Leverage public datasets like Wikipedia or Open Educational Resources (https://www.oercommons.org).
    - Explore nltk or spacy for advanced NLP if needed (e.g., parsing user inputs).
- Learning Support:
    - Django tutorials: https://docs.djangoproject.com/en/stable/
    - React tutorials: https://reactjs.org/docs/getting-started.html
    - Hugging Face tutorials: https://huggingface.co/docs/transformers
    - Vercel deployment guide: https://vercel.com/docs
- Portfolio Value: This project showcases full-stack development (Django, React), AI integration, and deployment skills, making it a strong addition to their portfolio.