Mathematical database development

programming languages and frameworks proposed :

Server Language Node.js, Database Neo4j, Express.js for routing and middleware management…

Server Language: Node.js, D3.js

Node.js is a strong choice for the backend for several reasons:

- **Real-time capabilities**: Ideal for applications requiring live updates, such as interactive knowledge graphs.
- **Rich ecosystem**: Easily integrates with D3.js and has numerous libraries for handling AI-related tasks and APIs.
- **Scalability**: Well-suited for high-concurrency scenarios.
- **Unified language**: If we're using JavaScript for both front-end (D3.js) and back-end (Node.js), it simplifies development and debugging.
- Framework suggestion: **Express.js** for routing and middleware management.

https://observablehq.com/@d3/gallery?utm_source=d3js-org&utm_medium=hero&utm_campaign=try-observable

Database: Neo4j for knowledge graph data

- **Neo4j** is a graph database that aligns naturally with the requirements of a knowledge graph:
- **Graph representation**: Designed to manage relationships between entities efficiently, making it ideal for your knowledge map.
- **Powerful query language**: Cypher query language allows intuitive querying of relationships and data traversal.
- **AI integration**: Supports integration with machine learning frameworks and pipelines.
- **Scalability**: Handles large datasets of interconnected nodes effectively
- **User-Friendly Tools**: Neo4j Desktop, Bloom, and Browser simplify development.

https://neo4j.com/pricing/

Use Cases for Knowledge Graphs in Neo4j

Enterprise Knowledge Graphs:

- Link corporate data for unified insights (e.g., customer 360 views, product catalogs).

Fraud Detection:

- Detect fraud patterns by analyzing relationships in financial networks.

Recommendation Systems:

- Recommend products, movies, or services based on user behavior and preferences.

Healthcare and Genomics:

- Model complex biological networks for drug discovery or disease mapping.

Semantic Search and NLP:

- Enable intelligent search by understanding the relationships between terms.

Database: PostgreSQL,...

Use a **relational database (PostgreSQL,...)** for managing other structured data (e.g., user profiles, question bank metadata).

Advantages of this Stack

- **Efficiency**: Neo4j excels at managing graph structures, crucial for the knowledge map.
- **Flexibility**: Node.js allows seamless integration with front-end technologies like D3.js and supports high levels of customization.
- **Future-proof**: Scalable architecture that can handle an increasing number of users and resources.
- **AI-readiness**: Python can complement this stack for advanced AI tasks via microservices.

Core Functions of the Knowledge Graph Engine

Concept Mapping:

- **Nodes**: Represent individual mathematical concepts, such as "Quadratic Equations" or "Pythagoras Theorem."
- **Edges**: Represent relationships between concepts, such as "is prerequisite of," "is related to," or "is part of."

Relationship Management:

- Dynamically define relationships (e.g., hierarchical, sequential, associative) between concepts.
- Handle cyclical and non-linear relationships to reflect real-world dependencies in math.

Dynamic Updates:

- Allow teachers to add, remove, or update concepts and relationships in real-time.
- Automatically update dependent nodes when changes are made (e.g., adding a new prerequisite).

Core Functions of the Knowledge Graph Engine

Visualization:

- Provide an intuitive, interactive interface for teachers and students to explore concepts and relationships.
- Highlight related concepts when a specific node is selected to provide context.

Querying:

- Enable advanced search and query functionality (e.g., “Find all prerequisites for Calculus” or “Show all topics related to Geometry”).
- Support pathfinding algorithms to identify shortest or most relevant learning paths.

Integration with AI Systems:

- Use AI to infer new relationships or dependencies based on user interactions, performance data, or predefined rules.
- Suggest relevant topics for preview or deeper exploration.

Data Persistence and Retrieval:

- Store the graph structure efficiently in a graph database (e.g., Neo4j) and provide fast retrieval for user queries.

Support for Mapping and Organizing Mathematical Concepts

Structured Knowledge Representation:

- Organizes concepts into a hierarchical or networked structure that reflects educational standards and logical dependencies.

Curriculum Mapping:

- Aligns nodes and relationships with curriculum standards for junior middle school mathematics.
- Includes metadata for each concept (e.g., difficulty level, associated question types, textbook references).

Customized Learning Paths:

- Identifies personalized learning paths for students by analyzing the graph and their progress.
- Suggests the next set of topics based on prerequisite knowledge and gaps identified through assessments.

Support for Mapping and Organizing Mathematical Concepts

Resource Linking:

- Associates resources (e.g., exercises, video tutorials, example problems) with corresponding nodes.
- Ensures that resources are contextually relevant and immediately accessible.

Teacher Support:

- Helps teachers design lesson plans by showing an optimized sequence of concepts.
- Highlights connections to reinforce related topics during teaching.

Real-Time Feedback:

- Tracks user interactions (e.g., frequently revisited nodes, problematic relationships) to refine the graph structure over time.
- Enables real-time adjustments to improve clarity or address user feedback.

Practical Example

- If a student struggles with "Quadratic Equations," the graph engine could:
- Identify prerequisite topics (e.g., "Factoring" or "Polynomials").
- Suggest related resources like videos or exercises to strengthen weak areas.
- Visualize connections between "Quadratic Equations" and advanced topics like "Graphing Parabolas," encouraging progression once foundational understanding is achieved.

Process and integrate the data integration part

To process and integrate data containing **images** and **text** while ensuring the accurate extraction of each knowledge point, a **multi-step approach** involving advanced processing techniques, tools, and AI models is necessary. Here's how this can be achieved:

Data Ingestion

- **Textual Data**:
  - Use **OCR (Optical Character Recognition)** tools (e.g., Tesseract, AWS Textract, Google Vision API) to extract text from scanned documents, textbooks, or handwritten notes.
  - Extract structured content from digital files (e.g., PDFs, Word documents) using libraries like **PyPDF2** or **Apache Tika**.
- **Images**:
  - For diagrams, formulas, or graphs, use **image processing tools** to detect and classify visual elements:
    - Leverage **deep learning-based models** (e.g., YOLO, OpenCV) to identify mathematical symbols, graphs, or figures.
    - Convert visual data into structured formats like LaTeX for equations or vector representations for graphs.

Process and integrate the data integration part

Preprocessing

- **Text Preprocessing**:
  - Tokenization and stemming to break down sentences into knowledge points.
  - Named Entity Recognition (NER) to identify specific mathematical terms, concepts, and relationships.
  - Context analysis using natural language processing (NLP) models like **SpaCy** or **transformers** (BERT, GPT).
- **Image Preprocessing**:
  - Enhance image quality (e.g., de-skewing, denoising).
  - Segment images into parts (e.g., separating text from diagrams or multiple-choice questions).

Process and integrate the data integration part

Knowledge Extraction

- **Text Extraction**:
  - Identify mathematical knowledge points (e.g., topics, theorems, rules) by matching against predefined taxonomies or using AI-based classification.
  - Extract relationships between knowledge points using dependency parsing or co-occurrence analysis.
- **Image Analysis**:
  - Classify diagrams, graphs, or problem setups using trained **image classification models**.
  - Extract embedded equations or annotations using a combination of OCR and formula recognition (e.g., MathPix API).

Process and integrate the data integration part

Data Structuring and Annotation

- **Mapping to Knowledge Graph**:
  - Match extracted knowledge points and their relationships to the predefined ontology of the knowledge graph.
  - Annotate concepts with metadata such as difficulty level, type (e.g., theorem, formula), and references.
- **Validation**:
  - Cross-check extracted data with curriculum standards or subject matter expert (SME) input to ensure accuracy.

Process and integrate the data integration part

Integration and Storage

- **Database Integration**:
  - Store structured data in a graph database (e.g., Neo4j) for **relationships** and a relational database (e.g., PostgreSQL) for **metadata**.
  - Link extracted images and text to corresponding knowledge graph nodes.
- **Indexing for Efficient Retrieval**:
  - Use search frameworks like **Elasticsearch** to enable fast querying and access to resources.

Process and integrate the data integration part

Ensuring Data Accuracy

- **Quality Assurance**:
  - Automated QA pipelines to check for inconsistencies or mismatches in extracted data.
  - Periodic human validation by subject matter experts (SMEs).
  - A **Subject Matter Expert (SME)** is a professional with deep knowledge, skills, and expertise in a specific area, field, or industry. SMEs are often consulted for their specialized insights and are critical in decision-making, project development, and ensuring the quality of work in their domain.
- **AI Model Fine-Tuning**:
  - Use feedback loops to refine the performance of AI models for text and image extraction based on errors detected during validation.

Process and integrate the data integration part

Tools and Technologies (Summary)

- **Text Processing**:
  - NLP libraries: SpaCy, Hugging Face Transformers, NLTK.
- **Image Processing**:
  - OCR: Tesseract, MathPix API.
  - Image Classification: TensorFlow, PyTorch, OpenCV.
- **Database**:
  - Neo4j for knowledge graph storage.
  - PostgreSQL or MongoDB for structured metadata and images.

Example Workflow

Input: A scanned textbook page with a diagram of a parabola and accompanying text.

Extraction:

- Textual content is extracted using OCR and NLP to identify the topic ("Graphing Quadratic Equations").
- The parabola diagram is classified and stored with metadata linking it to "Quadratic Equations."

Integration:

- Both text and image data are integrated into the knowledge graph, with edges denoting relationships (e.g., "is an example of" or "is derived from").

Output: A structured, searchable node in the knowledge graph for "Graphing Quadratic Equations," enriched with textual explanations and visual resources

This approach ensures **completeness**, **accuracy**, and **structured integration** of data for effective utilization in the platform.

Project using the Agile methodology, with weekly deployments on the Staging platform.

3 platforms :

- Development platform (managed by me)
- Staging platform (managed and/or used by you, for testing purposes)
- Production platform (managed and/or used by you)

What do I expect from you ?

- A precise documentation of the "functionalities, formulas, ..." you want to appear in order of priority (I will be able to establish a budget for the tasks to be completed).

- Do you provide the Staging environment (where you test the solution before deployment), Production environment ?