I want to create a website that gamifys Data Structures and Algorithms. I'm not sure about the tech stack, but I do want to use shadcn, with a light/dark mode. The rest of the frontend, I will leave it to you. I do want to have a login feature (including gmail login). Lets brainstorm on the rest. What are somethings I should consider, features, etc..

Architected comprehensive brainstorming framework for gamified DSA platform.

This is an exciting project! Gamifying DSA is a great way to make learning more engaging. Let me help you think through the key considerations, tech stack, and features.

Tech Stack Recommendations

Frontend:

- **Next.js 14** (App Router) - Great for SSR, routing, and works seamlessly with shadcn
- **shadcn/ui** + **Tailwind CSS** - As you requested
- **Framer Motion** - For smooth animations (important for gamification feedback)
- **Monaco Editor** or **CodeMirror** - For code editing challenges

Backend/Auth:

- **NextAuth.js** (now Auth.js) - Handles Gmail/OAuth easily, integrates perfectly with Next.js
- **Supabase** or **Firebase** - Great for real-time features, auth, and database
- Alternatively: **Prisma + PostgreSQL** if you want more control

Key Features to Consider

1. **Core Learning Mechanics**
   - **Interactive visualizations** - Animate how algorithms work (sorting, tree traversals, graph algorithms)
   - **Code challenges** with test cases and instant feedback
   - **Multiple difficulty levels** (Easy → Medium → Hard → Expert)
   - **Hints system** - Progressive hints that cost "coins" or affect score

2. **Gamification Elements**
   - **XP & Leveling system** - Earn XP for completing challenges
   - **Streaks** - Daily login/practice streaks (like Duolingo)
   - **Badges/Achievements** - "First Binary Tree", "Sorting Master", "100 Day Streak"
   - **Leaderboards** - Global, friends, weekly
   - **Lives/Hearts system** - Limited attempts that regenerate
   - **Coins/Currency** - Earn and spend on hints, cosmetics, unlocks

3. **Progression System**
   - **Skill trees** - Unlock advanced topics by mastering prerequisites
   - **Learning paths** - "Interview Prep", "Competitive Programming", "Fundamentals"
   - **Chapters/Worlds** - Theme different data structures (Array Kingdom, Tree Forest, Graph Galaxy)

4. **Social Features**
   - **Friends system** - Challenge friends, compare progress
   - **Multiplayer battles** - Race to solve problems
   - **Study groups/Clubs**

- **Discussion forums** per problem

5. **Content Types**
   - **Interactive lessons** - Learn concepts before challenges
   - **Flashcards** - For Big-O notation, terminology
   - **Quizzes** - Multiple choice for theory
   - **Coding challenges** - Write actual code
   - **Debugging challenges** - Find the bug
   - **Fill-in-the-blank** - Complete the algorithm

Questions to Help Define Scope

1. **Target audience?**
   - Complete beginners?
   - Interview prep (LeetCode alternative)?
   - Computer science students?
2. **Code execution?**
   - Will users run code? (Requires backend sandbox like Judge0)
   - Or just conceptual/visual learning?
3. **Content creation?**
   - Will you create all problems, or allow user-generated content?
4. **Monetization?**
   - Free with ads?
   - Freemium (limited problems free)?
   - Subscription?
5. **Mobile support?**
   - Responsive web only?
   - Native app later?

MVP Feature Set (Suggested)

For a first version, I'd suggest:

1. ☐ Auth (email + Gmail)
2. ☐ Dark/light mode
3. ☐ 3-5 data structure topics with interactive visualizations
4. ☐ 10-20 coding challenges with test cases
5. ☐ XP system + basic leveling
6. ☐ Daily streaks
7. ☐ Progress tracking dashboard
8. ☐ Basic leaderboard

What aspects resonate most with you? Would you like me to dive deeper into any of these areas, or should we start prototyping some UI components?

I want to be able to have/add custom questions, I suppose I would need a schema on how questions/answer need to be formatted. Also, I want different question types like drag and drop, fill in the blank, and others.

TB

Architected flexible schema supporting multiple question types and metadata.

Excellent thinking! A flexible question schema is crucial for scalability. Let me help you design a comprehensive system.

Question Types for DSA

Type	**Best For**	**Example**
Multiple Choice	Concepts, Big-O analysis	"What's the time complexity of binary search?"
Fill in the Blank	Code completion, syntax	`for (int i = 0; i < n; _____)`
Drag & Drop - Ordering	Algorithm steps, sorting	"Arrange these steps for QuickSort"
Drag & Drop - Matching	Matching concepts	Match data structure → use case
Drag & Drop - Code Blocks	Building algorithms	Arrange code blocks to form merge sort
Code Writing	Full implementation	"Implement a function to reverse a linked list"
Debugging	Find/fix errors	"This BFS has a bug. Fix it."
Visualization Interaction	Understanding operations	"Click where the next element inserts in this BST"
True/False	Quick concept checks	"A stack is FIFO"
Parsons Problems	Code ordering (no distractors)	Reorder scrambled lines of code

Proposed Schema Design

Here's a flexible TypeScript/JSON schema that supports all question types:

typescript

```typescript
// Base question interface
interface BaseQuestion {
  id: string;
  type: QuestionType;
  title: string;
  description: string;          // Markdown supported
  difficulty: 'beginner' | 'easy' | 'medium' | 'hard' | 'expert';
  topic: string[];              // ['arrays', 'sorting', 'quicksort']
  tags: string[];               // ['interview', 'google', 'recursion']
  xpReward: number;
  timeLimit?: number;           // seconds (optional)
  hints: Hint[];
  explanation: string;          // Shown after answer - markdown
  createdBy: string;            // user ID or 'system'
  createdAt: Date;
  isPublic: boolean;
  metadata?: Record<string, any>;
}

interface Hint {
  id: string;
  text: string;
  xpPenalty: number;            // XP deducted for using hint
  order: number;
}

type QuestionType =
  | 'multiple_choice'
  | 'multi select'
  | 'fill_blank'
  | 'drag_order'
  | 'drag_match'
  | 'drag_code_blocks'
  | 'code_writing'
  | 'debugging'
  | 'true_false'
  | 'visualization_click'
  | 'parsons';

// ============ SPECIFIC QUESTION TYPES ============

interface MultipleChoiceQuestion extends BaseQuestion {
  type: 'multiple_choice';
  content: {
    question: string;
    options: {
      id: string;
      text: string;              // Can include code blocks
      isCorrect: boolean;
    }[];
    shuffleOptions: boolean;
  };
}

interface MultiSelectQuestion extends BaseQuestion {
  type: 'multi_select';
  content: {
    question: string;
    instruction: string;         // "Select all that apply"
    options: {
      id: string;
      text: string;
      isCorrect: boolean;
    }[];
    shuffleOptions: boolean;
    partialCredit: boolean;      // Award partial points?
  };
}

interface FillBlankQuestion extends BaseQuestion {
  type: 'fill_blank';
  content: {
    // Text with blanks marked as {{blank_id}}
    template: string;            // "for(int i = {{b1}}; i < {{b2}}; i++)"
    blanks: {
      id: string;                // "b1"
      acceptedAnswers: string[]; // ["0", "start"] - multiple valid answers
      caseSensitive: boolean;
      placeholder?: string;
    }[];
    language?: string;           // For syntax highlighting
  };
}

interface DragOrderQuestion extends BaseQuestion {
  type: 'drag_order';
  content: {
    instruction: string;         // "Order these steps for DFS"
    items: {
      id: string;
      text: string;              // Can be code or text
      correctPosition: number;   // 0-indexed
    }[];
    includeDistractors: boolean; // Wrong items mixed in
    distractors?: {
      id: string;
      text: string;
    }[];
  };
}

interface DragMatchQuestion extends BaseQuestion {
  type: 'drag_match';
  content: {
    instruction: string;
    leftItems: {
      id: string;
      text: string;
      matchId: string;           // ID of correct right item
    }[];
    rightItems: {
      id: string;
      text: string;
    }[];
    // Extra right items = distractors
  };
}

interface DragCodeBlocksQuestion extends BaseQuestion {
  type: 'drag_code_blocks';
  content: {
    instruction: string;
    language: string;
    blocks: {
      id: string;
      code: string;
      correctPosition: number;
      indentLevel: number;       // For proper code formatting
    }[];
    distractorBlocks?: {
      id: string;
      code: string;
    }[];
  };
}

interface CodeWritingQuestion extends BaseQuestion {
  type: 'code_writing';
  content: {
    prompt: string;
    starterCode: string;
    language: string;
    testCases: {
      id: string;
      input: string;
      expectedOutput: string;
      isHidden: boolean;         // Hidden tests prevent hardcoding
      explanation?: string;
    }[];
    constraints?: string[];      // ["1 <= n <= 10^5", "O(n log n) expected"]
    solutionCode?: string;       // For explanation
    allowedLanguages?: string[]; // If multiple languages supported
  };
}

interface DebuggingQuestion extends BaseQuestion {
  type: 'debugging';
  content: {
    prompt: string;
    buggyCode: string;
    language: string;
    bugs: {
      lineNumber: number;
      bugDescription: string;    // For explanation
      correctCode: string;       // What it should be
    }[];
    testCases: {
      input: string;
      expectedOutput: string;
    }[];
  };
}

interface TrueFalseQuestion extends BaseQuestion {
  type: 'true_false';
  content: {
    statement: string;
    isTrue: boolean;
  };
}

interface VisualizationClickQuestion extends BaseQuestion {
  type: 'visualization_click';
  content: {
    instruction: string;          // "Click where 15 would be inserted"
    visualizationType: 'bst' | 'heap' | 'array' | 'linked_list' | 'graph';
    initialData: any;             // Structure-specific data
    correctClickTargets: string[]; // IDs of correct nodes/positions
    allowMultipleClicks: boolean;
  };
}

interface ParsonsQuestion extends BaseQuestion {
  type: 'parsons';
  content: {
    instruction: string;
    language: string;
    codeLines: {
      id: string;
      code: string;
      correctPosition: number;
      correctIndent: number;
    }[];
    // No distractors - pure reordering + indentation
  };
}

// ============ UNION TYPE ============
type Question =
  | MultipleChoiceQuestion
  | MultiSelectQuestion
  | FillBlankQuestion
  | DragOrderQuestion
  | DragMatchQuestion
  | DragCodeBlocksQuestion
  | CodeWritingQuestion
  | DebuggingQuestion
  | TrueFalseQuestion
  | VisualizationClickQuestion
  | ParsonsQuestion;
```

Example Questions (JSON)

Fill in the Blank

json

```json
{
  "id": "fib-001",
  "type": "fill_blank",
  "title": "Complete the Loop",
  "description": "Fill in the blanks to complete this array traversal.",
  "difficulty": "beginner",
  "topic": ["arrays", "loops"],
  "tags": ["fundamentals"],
  "xpReward": 10,
  "hints": [
    { "id": "h1", "text": "Arrays are 0-indexed", "xpPenalty": 2, "order": 1 }
  ],
  "explanation": "Arrays start at index 0 and go to length-1...",
  "createdBy": "system",
  "isPublic": true,
  "content": {
    "template": "for (int i = {{b1}}; i < arr.{{b2}}; {{b3}}) {\n  console.log(arr[i]);\n}",
    "blanks": [
      { "id": "b1", "acceptedAnswers": ["0"], "caseSensitive": false, "placeholder": "start" },
      { "id": "b2", "acceptedAnswers": ["length"], "caseSensitive": true, "placeholder": "?" },
      { "id": "b3", "acceptedAnswers": ["i++", "++i", "i+=1", "i = i + 1"], "caseSensitive": false, "pla
    ],
    "language": "javascript"
  }
}
```

Drag & Drop Ordering

json

```json
{
  "id": "bfs-order-001",
  "type": "drag_order",
  "title": "BFS Algorithm Steps",
  "description": "Arrange the steps of Breadth-First Search in the correct order.",
  "difficulty": "medium",
  "topic": ["graphs", "bfs", "traversal"],
  "tags": ["interview", "algorithm-steps"],
  "xpReward": 25,
  "hints": [],
  "explanation": "BFS uses a queue to explore level by level...",
  "createdBy": "system",
  "isPublic": true,
  "content": {
    "instruction": "Drag the steps into the correct order for BFS traversal",
    "items": [
      { "id": "s1", "text": "Initialize a queue and enqueue the starting node", "correctPosition": 0 },
      { "id": "s2", "text": "Mark the starting node as visited", "correctPosition": 1 },
      { "id": "s3", "text": "While queue is not empty, dequeue a node", "correctPosition": 2 },
      { "id": "s4", "text": "Process the dequeued node", "correctPosition": 3 },
      { "id": "s5", "text": "Enqueue all unvisited neighbors and mark them visited", "correctPosition":
    ],
    "includeDistractors": true,
    "distractors": [
      { "id": "d1", "text": "Push the node onto a stack" },
      { "id": "d2", "text": "Recursively call the function" }
    ]
  }
}
```

Database Considerations

For storing this in a database (like Supabase/PostgreSQL):

```sql
CREATE TABLE questions (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  type VARCHAR(50) NOT NULL,
  title VARCHAR(255) NOT NULL,
  description TEXT,
  difficulty VARCHAR(20) NOT NULL,
  topic TEXT[] NOT NULL,
  tags TEXT[],
  xp_reward INTEGER DEFAULT 10,
  time_limit INTEGER,
  hints JSONB DEFAULT '[]',
  explanation TEXT,
  content JSONB NOT NULL,  -- Type-specific content stored as JSON
  created_by UUID REFERENCES users(id),
  created_at TIMESTAMP DEFAULT NOW(),
  is_public BOOLEAN DEFAULT false,
  metadata JSONB
);

-- Index for efficient queries
CREATE INDEX idx_questions_topic ON questions USING GIN(topic);
CREATE INDEX idx_questions_type ON questions(type);
CREATE INDEX idx_questions_difficulty ON questions(difficulty);
```

Admin/Question Creator UI

You'll also want a question creation interface. Should I help you think through:

1. **A question builder UI** - Form-based creation for each type?
2. **Bulk import** - Upload questions via JSON/CSV?
3. **Question validation** - Ensure required fields, test correct answers?
4. **Preview mode** - Test questions before publishing?

What would you like to explore next? I can also start building a prototype component for any of these question types!

All of the above

Architected comprehensive design spanning UI, validation, import, and preview functionality.

Let me create a comprehensive system for all of this. I'll start by checking the frontend design skill to ensure we build this properly.

Reading frontend design skill for best practices

Architected comprehensive question builder with validation and preview capabilities.

Now let me create a comprehensive Question Builder System with all the features you requested. This will be a full admin interface with form-based creation, bulk import, validation, and preview mode.

Creating comprehensive Question Builder System with all features

question-builder.jsx

Orchestrated output delivery and comprehensive documentation creation.

Now let me create a comprehensive documentation file for the schema and import/export formats:

4 steps

Presented 2 files

Assembled and delivered comprehensive materials.

I've created a comprehensive Question Builder System for you! Here's what's included:

What I Built

1. Question Builder UI (`question-builder.jsx`)

A complete React component with shadcn/ui and dark/light mode featuring:

Builder Tab:

- Visual question type selector (10 types with icons)
- Base info form (title, description, difficulty, XP, topics)
- Type-specific content editors for each question type
- Hints system with XP penalties
- Explanation field for post-answer learning

Import Tab:

- JSON paste area for bulk import
- Schema reference panel
- Validation feedback for import errors

- Support for single questions or arrays

Library Tab:

- View all saved questions
- Edit existing questions
- Delete questions
- Export all as JSON

Features:

- Real-time validation with errors and warnings
- Live preview mode to test questions
- Copy JSON for individual questions
- Dark/light mode toggle

2. Schema Documentation (`QUESTION_SCHEMA.md`)

Comprehensive docs covering:

- All 10 question types with TypeScript interfaces
- JSON examples for each type
- Validation rules (required fields, type-specific)
- Import/export formats (JSON & CSV)
- PostgreSQL database schema
- Best practices for writing questions
- Suggested API endpoints

Summary of Question Types

Type	**Interactive Element**
Multiple Choice	Radio buttons
Multi Select	Checkboxes
Fill in Blank	Text inputs in code
Drag Order	Sortable list
Drag Match	Connect pairs
Code Blocks	Arrange code pieces
Code Writing	Full code editor + tests
Debugging	Fix buggy code
True/False	Binary choice
Parsons	Order + indent code

Next Steps

Would you like me to:

1. **Build the student-facing question renderer** - The UI students see when answering questions
2. **Create the code execution sandbox** - For running code_writing questions
3. **Design the gamification system** - XP, levels, streaks, achievements
4. **Set up the authentication** - NextAuth with Gmail login
5. **Create visualization components** - For algorithm animations (sorting, trees, graphs)

Let me know which direction to go next!

Question builder
Code · JSX

Download

Question schema
Document · MD

Download

Download all

Reply...

Opus 4.5

Claude is AI and can make mistakes. Please double-check responses.