StruggleForum: Minimal Viable Product Specification

February 3, 2026

Contents

1 Introduction

This document defines the minimal viable product (MVP) specification for StruggleForum, including the core data models, API endpoints, permission rules, and session management requirements. The specification includes technology-specific implementation details using the chosen stack.

1.1 Technology stack

To implement the MVP for the StruggleForum the following technologies are used:

1. Postgres 16 for the DB
2. NextJS framework for backend and frontend
3. Prisma ORM to have easily written queries to the db using TS(JS).
4. ReactJS library for the frontend
5. TypeScript for more strict typing

2 Data Models

This section describes the core entities in the forum system.

2.1 User

- **Fields:**
  - `id`: String (UUID, default: uuid())
  - `username`: String (unique)
  - `email`: String (unique)
  - `password_hash`: String
  - `role`: Enum (USER, ADMIN) - default: USER
  - `created_at`: DateTime (default: now())
  - `updated_at`: DateTime (updatedAt)
- **Constraints:**
  - Username must be unique
  - Email must be unique and valid

Prisma Schema:

```
model User {
  id            String    @id @default(uuid())
  username      String    @unique
  email         String    @unique
  passwordHash  String    @map("password_hash")
  role          Role      @default(USER)
  createdAt     DateTime  @default(now()) @map("created_at")
  updatedAt     DateTime  @updatedAt @map("updated_at")

  sessions      Session[]
  posts         Post[]
  comments      Comment[]
  likes         Like[]
  notifications Notification[]

  @@map("users")
}

enum Role {
  USER
  ADMIN
}
```

TypeScript Type:

```
type User = {
  id: string;
  username: string;
  email: string;
  passwordHash: string;
  role: 'USER' | 'ADMIN';
  createdAt: Date;
  updatedAt: Date;
};

type UserPublic = Omit<User, 'passwordHash'>;
```

2.2 Session

- **Fields:**
  - `id`: String (UUID, default: uuid())
  - `user_id`: String (Foreign key to User)
  - `token`: String (unique)
  - `created_at`: DateTime (default: now())
  - `last_activity`: DateTime
  - `expires_at`: DateTime
- **Rules:**
  - Sessions expire after **2 hours of inactivity**
  - Last activity timestamp updates on each authenticated request

Prisma Schema:

```
model Session {
  id           String   @id @default(uuid())
  userId       String   @map("user_id")
  token        String   @unique
  createdAt    DateTime @default(now()) @map("created_at")
  lastActivity DateTime @map("last_activity")
  expiresAt    DateTime @map("expires_at")

  user User @relation(fields: [userId], references: [id], onDelete:
     Cascade)

  @@index([userId])
  @@index([token])
  @@map("sessions")
}
```

TypeScript Type:

```
type Session = {
  id: string;
  userId: string;
  token: string;
  createdAt: Date;
  lastActivity: Date;
  expiresAt: Date;
};
```

2.3 Post

- **Fields:**
  - `id`: String (UUID, default: uuid())
  - `author_id`: String (Foreign key to User)
  - `category_id`: String (Foreign key to Category)
  - `title`: String
  - `content`: String (Text)
  - `locked`: Boolean (default: false)
  - `created_at`: DateTime (default: now())
  - `updated_at`: DateTime (updatedAt)
- **Constraints:**
  - Title is required
  - Content is required
  - Category is required
- **Moderation:**
  - `locked`: Prevents new comments (admin only)

Prisma Schema:

```
model Post {
  id         String    @id @default(uuid())
  authorId   String    @map("author_id")
  categoryId String    @map("category_id")
  title      String
  content    String    @db.Text
  locked     Boolean   @default(false)
  createdAt  DateTime  @default(now()) @map("created_at")
  updatedAt  DateTime  @updatedAt @map("updated_at")

  author    User     @relation(fields: [authorId], references: [id],
     onDelete: Cascade)
  category  Category @relation(fields: [categoryId], references: [id],
     onDelete: Restrict)
  comments  Comment[]
  likes     Like[]

  @@index([authorId])
  @@index([categoryId])
  @@index([createdAt])
  @@map("posts")
}
```

TypeScript Types:

```
type Post = {
  id: string;
  authorId: string;
  categoryId: string;
  title: string;
  content: string;
  locked: boolean;
  createdAt: Date;
  updatedAt: Date;
};

type PostWithRelations = Post & {
  author: UserPublic;
  category: Category;
  _count: {
    likes: number;
    comments: number;
  };
};
```

2.4 Comment

- **Fields:**
  - `id`: String (UUID, default: uuid())
  - `post_id`: String (Foreign key to Post)
  - `author_id`: String (Foreign key to User)
  - `parent_id`: String (Foreign key to Comment, optional)
  - `content`: String (Text)

  - `depth`: Int (default: 0)
  - `locked`: Boolean (default: false)
  - `created_at`: DateTime (default: now())
  - `updated_at`: DateTime (updatedAt)

- **Constraints:**
  - Content is required

- **Nested Replies:**
  - Comments can reply to other comments via `parent_id`
  - Top-level comments have `parent_id = null` and `depth = 0`
  - Each reply increments the depth level

- **Moderation:**
  - `locked`: Prevents replies to this comment (admin only)

Prisma Schema:

```
model Comment {
  id        String    @id @default(uuid())
  postId    String    @map("post_id")
  authorId  String    @map("author_id")
  parentId  String?   @map("parent_id")
  content   String    @db.Text
  depth     Int       @default(0)
  locked    Boolean   @default(false)
  createdAt DateTime  @default(now()) @map("created_at")
  updatedAt DateTime  @updatedAt @map("updated_at")

  post    Post     @relation(fields: [postId], references: [id],
     onDelete: Cascade)
  author  User     @relation(fields: [authorId], references: [id],
     onDelete: Cascade)
  parent  Comment? @relation("CommentReplies", fields: [parentId],
     references: [id], onDelete: Cascade)
  replies Comment[] @relation("CommentReplies")
  likes   Like[]

  @@index([postId])
  @@index([authorId])
  @@index([parentId])
  @@map("comments")
}
```

TypeScript Types:

```
type Comment = {
  id: string;
  postId: string;
  authorId: string;
  parentId: string | null;
  content: string;
  depth: number;
  locked: boolean;
  createdAt: Date;
  updatedAt: Date;
};

type CommentWithAuthor = Comment & {
  author: UserPublic;
  _count: {
    likes: number;
  };
};

type CommentWithReplies = CommentWithAuthor & {
  replies: CommentWithAuthor[];
};
```

2.5 Category

- **Fields:**
  - `id`: String (UUID, default: uuid())

  - `name`: String (unique)
  - `description`: String (optional)
  - `created_at`: DateTime (default: now())

- **Relationships:**
  - One-to-many relationship with Post
  - Each post belongs to exactly one category

- **Constraints:**
  - Name must be unique

Prisma Schema:

```
model Category {
  id          String   @id @default(uuid())
  name        String   @unique
  description String?
  createdAt   DateTime @default(now()) @map("created_at")

  posts Post[]

  @@map("categories")
}
```

TypeScript Type:

```
type Category = {
  id: string;
  name: string;
  description: string | null;
  createdAt: Date;
};

type CategoryWithCount = Category & {
  _count: {
    posts: number;
  };
};
```

2.6 Notification

- **Fields:**
  - `id`: String (UUID, default: uuid())
  - `user_id`: String (Foreign key to User)
  - `type`: Enum (COMMENT, LIKE, MENTION)
  - `content`: String
  - `reference_id`: String (optional)
  - `is_read`: Boolean (default: false)
  - `created_at`: DateTime (default: now())

Prisma Schema:

```
model Notification {
  id          String           @id @default(uuid())
  userId      String           @map("user_id")
  type        NotificationType
  content     String
  referenceId String?          @map("reference_id")
  isRead      Boolean          @default(false) @map("is_read")
  createdAt   DateTime         @default(now()) @map("created_at")

  user User @relation(fields: [userId], references: [id], onDelete:
     Cascade)

  @@index([userId])
  @@index([isRead])
  @@map("notifications")
}

enum NotificationType {
  COMMENT
  LIKE
  MENTION
}
```

TypeScript Type:

```
type Notification = {
  id: string;
  userId: string;
  type: 'COMMENT' | 'LIKE' | 'MENTION';
  content: string;
  referenceId: string | null;
  isRead: boolean;
  createdAt: Date;
};
```

2.7 Like

- **Fields:**
  - `id`: String (UUID, default: uuid())
  - `user_id`: String (Foreign key to User)
  - `post_id`: String (Foreign key to Post, optional)
  - `comment_id`: String (Foreign key to Comment, optional)
  - `created_at`: DateTime (default: now())
- **Constraints:**
  - User can like an entity only once
  - Either postId or commentId must be set (not both)

Prisma Schema:

```
model Like {
  id        String   @id @default(uuid())
  userId    String   @map("user_id")
  postId    String?  @map("post_id")
  commentId String?  @map("comment_id")
  createdAt DateTime @default(now()) @map("created_at")

  user    User     @relation(fields: [userId], references: [id],
     onDelete: Cascade)
  post    Post?    @relation(fields: [postId], references: [id],
     onDelete: Cascade)
  comment Comment? @relation(fields: [commentId], references: [id],
     onDelete: Cascade)

  @@unique([userId, postId])
  @@unique([userId, commentId])
  @@index([userId])
  @@index([postId])
  @@index([commentId])
  @@map("likes")
}
```

TypeScript Type:

```
type Like = {
  id: string;
  userId: string;
  postId: string | null;
  commentId: string | null;
  createdAt: Date;
};
```

3 API Endpoints

This section defines the required endpoints for the forum API.

3.1 Authentication Endpoints

3.1.1 POST /api/auth/register

File: `app/api/auth/register/route.ts`

Request:

```
{
  "username": "string",
  "email": "string",
  "password": "string"
}
```

Response (201):

```
{
  "user": {
    "id": "uuid",
    "username": "string",
    "email": "string",
```

```
    "role": "USER",
    "createdAt": "2026-02-02T10:00:00.000Z"
  },
  "token": "string"
}
```

Next.js Implementation Example:

```
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/lib/prisma';
import bcrypt from 'bcryptjs';
import { generateToken } from '@/lib/auth';

export async function POST(req: NextRequest) {
  const { username, email, password } = await req.json();

  // Hash password
  const passwordHash = await bcrypt.hash(password, 10);

  // Create user
  const user = await prisma.user.create({
    data: { username, email, passwordHash },
  });

  // Create session
  const token = generateToken();
  await prisma.session.create({
    data: {
      userId: user.id,
      token,
      lastActivity: new Date(),
      expiresAt: new Date(Date.now() + 2 * 60 * 60 * 1000),
    },
  });

  const { passwordHash: _, ...userPublic } = user;
  return NextResponse.json({ user: userPublic, token }, { status: 201
     });
}
```

3.1.2 POST /api/auth/login

Request:

```
{
  "email": "string",
  "password": "string"
}
```

Response (200):

```
{
  "user": {
    "id": "uuid",
    "username": "string",
    "email": "string",
    "role": "string"
  },
  "token": "string"
}
```

3.1.3 POST /api/auth/logout

Headers: Authorization: Bearer {token}

Response (200):

```
{
  "message": "Logged out successfully"
}
```

3.2 Post Endpoints

3.2.1 GET /api/posts

Query Parameters:

- `page`: Page number (default: 1)
- `limit`: Items per page (default: 20)
- `category`: Filter by category ID

Response (200):

```
{
  "posts": [
    {
      "id": "uuid",
      "title": "string",
      "content": "string",
      "author": { "id": "uuid", "username": "string" },
      "category": { "id": "uuid", "name": "string" },
      "like_count": 0,
      "comment_count": 0,
      "created_at": "timestamp",
      "updated_at": "timestamp"
    }
  ],
  "pagination": {
    "page": 1,
    "limit": 20,
    "total": 100
  }
}
```

3.2.2 GET /api/posts/:id

Response (200):

```
{
  "id": "uuid",
  "title": "string",
  "content": "string",
  "author": { "id": "uuid", "username": "string" },
  "category": { "id": "uuid", "name": "string" },
  "like_count": 0,
  "comment_count": 0,
  "created_at": "timestamp",
  "updated_at": "timestamp"
}
```

3.2.3 POST /api/posts

Headers: Authorization: Bearer {token}

Request:

```
{
  "title": "string",
  "content": "string",
  "categoryId": "uuid"
}
```

Response (201):

```
{
  "id": "uuid",
  "title": "string",
  "content": "string",
  "author": { "id": "uuid", "username": "string" },
  "category": { "id": "uuid", "name": "string" },
  "created_at": "timestamp",
  "updated_at": "timestamp"
}
```

3.2.4 PUT /api/posts/:id

Headers: Authorization: Bearer {token}

Request:

```
{
  "title": "string",
  "content": "string",
  "categoryId": "uuid"
}
```

Response (200):

```
{
  "id": "uuid",
  "title": "string",
  "content": "string",
  "category": { "id": "uuid", "name": "string" },
  "updated_at": "timestamp"
}
```

3.2.5 DELETE /api/posts/:id

Headers: Authorization: Bearer {token}

Response (200):

```
{
  "message": "Post deleted successfully"
}
```

3.3 Comment Endpoints

3.3.1 GET /api/posts/:id/comments

Response (200):

```
{
  "comments": [
    {
      "id": "uuid",
      "content": "string",
      "author": { "id": "uuid", "username": "string" },
      "parentId": null,
      "depth": 0,
      "like_count": 0,
      "created_at": "timestamp",
      "updated_at": "timestamp",
      "replies": [
        {
          "id": "uuid",
          "content": "string",
          "author": { "id": "uuid", "username": "string" },
          "parentId": "parent-uuid",
          "depth": 1,
          "like_count": 0,
          "created_at": "timestamp",
          "updated_at": "timestamp",
          "replies": []
        }
      ]
    }
  ]
}
```

3.3.2 POST /api/posts/:id/comments

Headers: Authorization: Bearer {token}

Request:

```
{
  "content": "string",
  "parentId": "uuid" // Optional: for nested replies
}
```

Response (201):

```
{
  "id": "uuid",
  "content": "string",
  "author": { "id": "uuid", "username": "string" },
  "parentId": "uuid", // null for top-level comments
  "depth": 0,
  "created_at": "timestamp"
}
```

3.3.3 PUT /api/comments/:id

Headers: Authorization: Bearer {token}

Request:

```
{
  "content": "string"
}
```

Response (200):

```
{
  "id": "uuid",
  "content": "string",
  "updated_at": "timestamp"
}
```

3.3.4 DELETE /api/comments/:id

Headers: Authorization: Bearer {token}
Response (200):

```
{
  "message": "Comment deleted successfully"
}
```

3.4 Like Endpoints

3.4.1 POST /api/likes

Headers: Authorization: Bearer {token}
Request:

```
{
  "target_type": "post|comment",
  "target_id": "uuid"
}
```

Response (201):

```
{
  "id": "uuid",
  "target_type": "string",
  "target_id": "uuid",
  "created_at": "timestamp"
}
```

3.4.2 DELETE /api/likes/:id

Headers: Authorization: Bearer {token}
Response (200):

```
{
  "message": "Like removed successfully"
}
```

3.5 Notification Endpoints

3.5.1 GET /api/notifications

Headers: Authorization: Bearer {token}
Response (200):

```
{
  "notifications": [
    {
      "id": "uuid",
      "type": "string",
      "content": "string",
      "reference_id": "uuid",
      "is_read": false,
      "created_at": "timestamp"
    }
  ]
}
```

3.5.2 PUT /api/notifications/:id/read

Headers: Authorization: Bearer {token}

Response (200):

```
{
  "id": "uuid",
  "is_read": true
}
```

3.6 Category Endpoints

3.6.1 GET /api/categories

Response (200):

```
{
  "categories": [
    {
      "id": "uuid",
      "name": "string",
      "description": "string",
      "post_count": 0
    }
  ]
}
```

3.6.2 POST /api/categories

Headers: Authorization: Bearer {token} (Admin only)

Request:

```
{
  "name": "string",
  "description": "string"
}
```

Response (201):

```
{
  "id": "uuid",
  "name": "string",
  "description": "string",
  "created_at": "timestamp"
```

```
}
```

3.7 Admin/Moderation Endpoints

3.7.1 POST /api/admin/posts/:id/lock

Headers: Authorization: Bearer {token} (Admin only)
Response (200):

```
{
  "id": "uuid",
  "locked": true
}
```

3.7.2 POST /api/admin/posts/:id/unlock

Headers: Authorization: Bearer {token} (Admin only)
Response (200):

```
{
  "id": "uuid",
  "locked": false
}
```

3.7.3 POST /api/admin/comments/:id/lock

Headers: Authorization: Bearer {token} (Admin only)
Response (200):

```
{
  "id": "uuid",
  "locked": true
}
```

3.7.4 POST /api/admin/comments/:id/unlock

Headers: Authorization: Bearer {token} (Admin only)
Response (200):

```
{
  "id": "uuid",
  "locked": false
}
```

4 Permission Rules

This section defines the authorization rules for different user roles.

4.1 User Permissions

Regular users (`role: "user"`) can:

- View all posts and comments (read-only access)
- Create new posts

- Edit their own posts
- Delete their own posts
- Create comments on any post
- Edit their own comments
- Delete their own comments
- Like/unlike posts and comments
- View and manage their own notifications

Users **cannot**:

- Edit or delete posts created by other users
- Edit or delete comments created by other users
- Access or manage other users' notifications
- Perform administrative actions

4.2 Admin Permissions

Administrators (`role: "admin"`) have all user permissions plus:

- Edit any post (regardless of author)
- Delete any post
- Edit any comment
- Delete any comment
- Create, edit, and delete categories
- View all users
- Moderate content across the platform

4.3 Anonymous Access

Unauthenticated users can:

- View all posts (read-only)
- View all comments (read-only)
- View categories

Anonymous users **cannot**:

- Create, edit, or delete any content
- Like posts or comments
- Access notifications

5 Session Management

5.1 Session Expiry Rule

- Sessions expire after **2 hours of inactivity**
- The `last_activity` timestamp is updated on every authenticated API request
- Inactivity is defined as the time elapsed since the `last_activity` timestamp
- When a session expires:
  - The session token becomes invalid
  - Subsequent requests with the expired token return 401 Unauthorized
  - The user must log in again to obtain a new session token

5.2 Session Validation

On every authenticated request, the system must:

1. Verify the session token exists and is valid
2. Check if the session has expired (current time - last_activity ¿ 2 hours)
3. If valid: update the `last_activity` timestamp to current time
4. If expired: return 401 Unauthorized and invalidate the session

5.3 Next.js Middleware Implementation

File: `middleware.ts`

```
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/lib/prisma';

export async function middleware(req: NextRequest) {
  const token = req.headers.get('authorization')?.replace('Bearer ',
     '');

  if (!token) {
    return NextResponse.json(
      { error: { code: 'UNAUTHORIZED', message: 'No token provided' }
         },
      { status: 401 }
    );
  }

  const session = await prisma.session.findUnique({
    where: { token },
    include: { user: true },
  });

  if (!session) {
    return NextResponse.json(
      { error: { code: 'UNAUTHORIZED', message: 'Invalid token' } },
      { status: 401 }
    );
  }

  // Check if session expired (2 hours of inactivity)
  const twoHoursAgo = new Date(Date.now() - 2 * 60 * 60 * 1000);
  if (session.lastActivity < twoHoursAgo) {
    await prisma.session.delete({ where: { id: session.id } });
    return NextResponse.json(
      { error: { code: 'SESSION_EXPIRED', message: 'Session expired' }
         },
      { status: 401 }
    );
  }

  // Update last activity
  await prisma.session.update({
    where: { id: session.id },
    data: {
      lastActivity: new Date(),
      expiresAt: new Date(Date.now() + 2 * 60 * 60 * 1000),
    },
  });

  // Add user to request headers for route handlers
  const response = NextResponse.next();
  response.headers.set('x-user-id', session.userId);
  response.headers.set('x-user-role', session.user.role);
  return response;
}

export const config = {
  matcher: '/api/((?!auth/login|auth/register).*)'
};
```

6 Implementation Notes

6.1 Prisma Setup

File: `lib/prisma.ts`

```
import { PrismaClient } from '@prisma/client';

const globalForPrisma = global as unknown as { prisma: PrismaClient };

export const prisma = globalForPrisma.prisma || new PrismaClient();

if (process.env.NODE_ENV !== 'production') {
  globalForPrisma.prisma = prisma;
}
```

Environment Variables (.env):

```
DATABASE_URL="postgresql://user:password@localhost:5432/struggleforum?
   schema=public"
NEXT_PUBLIC_API_URL="http://localhost:3000"
```

Required npm packages:

```
npm install @prisma/client
npm install -D prisma
npm install bcryptjs
npm install -D @types/bcryptjs
```

Prisma Commands:

```
# Initialize Prisma
npx prisma init

# Create migration
npx prisma migrate dev --name init

# Generate Prisma Client
npx prisma generate

# Open Prisma Studio
npx prisma studio
```

6.2 Required Validations

- Email format validation on registration
- Password strength requirements (minimum 8 characters)
- Input sanitization to prevent XSS attacks
- Rate limiting on authentication endpoints

6.3 Error Responses

All error responses should follow this format:

```
{
  "error": {
    "code": "ERROR_CODE",
    "message": "Human-readable error message",
    "details": {}
  }
}
```

Common HTTP status codes:

- 400: Bad Request (validation errors)
- 401: Unauthorized (authentication required/failed)
- 403: Forbidden (insufficient permissions)
- 404: Not Found
- 500: Internal Server Error