Below is a backend-oriented view of your system, based on the requirements document + use cases + diagrams. I'll first summarize the domain model and flow (so the endpoints make sense), then list concrete endpoint groups with responsibilities, main backend operations, and typical responses you can use for task distribution.

---

1. Domain model & program flow (from system models)

1.1 Core entities & relationships

From the object model (page 23) and functional requirements: Topics, Participants, Teams, Events, 6-3-5 Sessions, Rounds, Ideas, AIArtifacts (suggestions/summaries), SessionLogs.

Conceptually:

- **User / Participant**
  - Single table `users` (with role field: EVENT_MANAGER / TEAM_LEADER / TEAM_MEMBER).
  - A "Participant" is a user assigned into an event/team; Event Managers are also users with elevated permissions.
- **Event**
  - Groups Topics, Teams, Sessions.
  - Event Manager owns/manages the event.
  - Each Participant can belong to **one team per event**.
- **Topic**
  - Belongs to an Event.
  - Fields: `title`, `description`, `status` (open/closed/archived). CRUD controlled by Event Manager.
- **Team**
  - Belongs to an Event.
  - Has one Team Leader (userId) and 5 other Team Members (6 total for a valid 6-3-5 session).

- **Session (6-3-5 brainstorm)**
  - Belongs to a Team and a Topic.
  - Has status: `PENDING`, `RUNNING`, `PAUSED`, `COMPLETED`.
  - Composed of 5 Rounds (for the 6-3-5 method).
- **Round**
  - Belongs to a Session.
  - Has `roundNumber` (1–5), `startTime`, `endTime`, `timerState`.
  - Holds all Idea submissions for that round.
- **Idea**
  - Belongs to exactly one Round, one Session, one Team, and one author (Participant).
  - Contains text + timestamps + optional AI annotations.
- **AIArtifact**
  - Type: `SUGGESTION` or `SUMMARY`.
  - Belongs to a Session (and maybe a Round).
  - Generated via ChatGPT; linked to the input ideas/ topic.
- **SessionLog**
  - Records events: session created, round started/ended, timer events, AI calls, idea submissions, role actions.

This matches the diagrams:

- Use cases cluster actions around Topics/Participants/Teams, Sessions, AI, Reporting.
- Sequence diagrams show controllers: `BrainstormingController`, `TeamController`, `SessionController`, `AIController`, `ReportController`, `AuthController`.

1.2 High-level flow (6-3-5 session lifecycle)

From scenarios & activity diagrams:

1. **Event configuration (Event Manager)**
   - Create Event → add Topics → register Participants → form Teams (+ assign TLs).

2. **Session setup (Team Leader)**
    - Select Topic + Team → create Session (status PENDING).
3. **Run 6-3-5**
    - TL (or EM) starts Round 1 → timer starts → each of 6 members submits exactly 3 ideas.
    - When timer expires or all submitted, system:
        - saves ideas,
        - "passes" each set of 3 ideas to next participant,
        - creates or activates next Round, repeat until 5 rounds.
4. **AI help (optional)**
    - During rounds: TL/EM call AI for idea suggestions given current topic + ideas.
    - After session: TL/EM call AI summarization for all session ideas.
5. **Reporting & export (Event Manager)**
    - View completed sessions, summaries, logs.
    - Export to PDF/CSV.

The endpoints below are structured against this flow so you can assign backend tasks per module.

---

2. API grouping

I'll group endpoints into the following logical modules:

1. Authentication & User Profile
2. Participants & Roles
3. Events, Topics & Teams
4. 6-3-5 Sessions & Rounds (core engine)
5. Ideas (submission & viewing)
6. AI Integration (ChatGPT)
7. Real-Time / Notifications
8. Reporting, Export & Audit

For each, I'll give:

- **Endpoint & Method**
- **Role access**
- **Request "loadout"** (main fields)
- **Backend responsibilities / operations**
- **Typical response payload**

You can refine naming and details during implementation.

---

3. Authentication & User Profile

These cover FR-700–702 and Scenario 10 (loginAuthentication).

3.1 POST /auth/register

- **Roles**: Public (no auth).
- **Use**: Self-registration as Participant (FR-102).
- **Body (example fields)**:
  - `email`, `password`, `fullName`, `organization`, `phone`, optionally `preferredRole` (default TEAM_MEMBER).
- **Backend responsibilities**
  - Validate input & uniqueness of email.
  - Create `User` with default role TEAM_MEMBER and status `PENDING` or `ACTIVE` depending on your policy.
  - Optionally send confirmation mail.
- **Response**
  - `201` with `{ id, email, fullName, role, status, token? }`.

3.2 POST /auth/login

- **Roles**: Public.
- **Use**: Authenticate and return JWT (FR-700).
- **Body**: `email`, `password`.

- **Backend responsibilities**
  - Verify credentials.
  - Issue JWT with embedded role + userId.
  - Record login event in SessionLog.
- **Response**
  - `200 { token, user: { id, role, fullName } }` or `401` error.

3.3 GET /auth/me

- **Roles**: Authenticated.
- **Use**: Get current user profile and roles.
- **Backend responsibilities**
  - Decode JWT, fetch from DB.
- **Response**
  - `200 { id, email, fullName, role, assignedTeams, events }`.

---

4. Participants & Roles

Covers FR-101–114, Scenario 2, User Registration / Management flow.

4.1 POST /participants

- **Roles**: EVENT_MANAGER
- **Use**: Register participant manually (FR-101).
- **Body**: `fullName`, `email`, `phone`, `role`, `eventId`.
- **Backend responsibilities**
  - Create User/Participant.
  - Optionally send invitation email.
- **Response**
  - `201 { id, fullName, role, status, eventId }`.

4.2 GET /participants

- **Roles**: EVENT_MANAGER. Limited view for TEAM_LEADER.
- **Use**: List participants; TL only sees own team members (FR-106–107).
- **Query params**: `eventId`, `teamId`, `role`, pagination.
- **Backend responsibilities**
  - Apply role-based filters.
- **Response**
  - `200` list of participants (fields filtered based on role).

4.3 GET /participants/{id}

- **Roles**: EVENT_MANAGER (full), TEAM_LEADER (members in own team), SELF (own record).
- **Use**: Get detailed participant info.
- **Response**: `200` participant details or `403` / `404`.

4.4 PATCH /participants/{id}

- **Roles**: EVENT_MANAGER.
- **Use**: Update contact info, role, status (FR-110).
- **Body**: Partial: `{ fullName?, phone?, role?, status? }`.
- **Responsibilities**
  - Validate role transitions (e.g., Team Leader must be member of a team).
- **Response**: `200` updated participant.

4.5 DELETE /participants/{id}

- **Roles**: EVENT_MANAGER.
- **Use**: Remove participant from event roster (FR-114).
- **Response**: `204`.

---

5. Events, Topics & Teams

5.1 Events

Although Events are not fully spelled as FRs, they're in the DB design and scenarios.

5.1.1 POST /events

- **Roles**: EVENT_MANAGER.
- **Use**: Create an event container.
- **Body**: `name`, `description`, `startDate`, `endDate`.
- **Responsibilities**: Insert event; set EM as owner.
- **Response**: `201` event.

5.1.2 GET /events, GET /events/{id}, PATCH, DELETE

- Standard CRUD for Event Manager to manage events; PATCH to update metadata, DELETE to archive/soft-delete.

5.2 Topics

FR-100, FR-105, FR-109, FR-113 and Scenario 1.

5.2.1 POST /events/{eventId}/topics

- **Roles**: EVENT_MANAGER.
- **Body**: `title`, `description`.
- **Responsibilities**
  - Create Topic linked to event with status `OPEN`.
- **Response**: `201` topic.

5.2.2 GET /events/{eventId}/topics

- **Roles**: Any authenticated (FR-105).
- **Use**: List topics in event.
- **Response**: list of `{ id, title, description, status }`.

5.2.3 GET /topics/{topicId}

- Same, with detail.

5.2.4 PATCH /topics/{topicId}

- **Roles**: EVENT_MANAGER.
- **Body**: any of `title`, `description`, `status`.
- **Responsibilities**: Update; maybe prevent changes after event closed.
- **Response**: `200` topic.

5.2.5 DELETE /topics/{topicId}

- **Roles**: EVENT_MANAGER.
- **Use**: Delete obsolete topic (FR-113).
- **Response**: `204`.

5.3 Teams

FR-103–115, Scenario 3, Team Creation & Member Assignment flow.

5.3.1 POST /events/{eventId}/teams

- **Roles**: EVENT_MANAGER or TEAM_LEADER (FR-103, FR-104).
- **Body**: `name`, `leaderId`, `memberIds[]`.
- **Responsibilities**
  - Validate all users belong to same event.
  - Enforce max 6 members (or at least 6 for 6-3-5 sessions).
- **Response**: `201` team.

5.3.2 GET /events/{eventId}/teams

- **Roles**: EVENT_MANAGER (all), TEAM_LEADER (own), TEAM_MEMBER (own team).
- **Use**: View all teams / own team composition (FR-107–108).
- **Response**: list with membership details.

5.3.3 GET /teams/{teamId}

- Detailed view.

5.3.4 PATCH /teams/{teamId}

- **Roles**: EVENT_MANAGER, TEAM_LEADER (within own team).
- **Body options**
  - `name`, `leaderId`, `addMembers[]`, `removeMembers[]`.
- **Responsibilities**
  - Update composition (FR-112).
  - Maintain invariant "participant belongs to only one team per event".
- **Response**: `200` updated team.

5.3.5 DELETE /teams/{teamId}

- **Roles**: EVENT_MANAGER.
- **Use**: Disband team (FR-115).
- **Response**: `204`.

5.3.6 POST /teams/{teamId}/members

- **Roles**: EVENT_MANAGER, TEAM_LEADER.
- **Use**: Add member(s) to team (FR-111).
- **Body**: `memberIds[]`.

5.3.7 DELETE /teams/{teamId}/members/{userId}

- **Roles**: EVENT_MANAGER, TEAM_LEADER.
- **Use**: Remove member from team.

---

6. 6-3-5 Sessions & Rounds (core engine)

These implement FR-200–210 and the 6-3-5 session flow diagrams.

6.1 POST /teams/{teamId}/sessions

- **Roles**: TEAM_LEADER (own team), EVENT_MANAGER (for any team).
- **Use**: Create a 6-3-5 session for a team. (Scenario 4.)
- **Body**: `topicId`, optional `name`, `roundCount` (default 5).

- **Responsibilities**
  - Check team has 6 active members (FR-200).
  - Create Session with status `PENDING`, rounds metadata if you pre-create 5 rounds.
- **Response**
  - `201 { sessionId, teamId, topicId, status, roundCount }`.

6.2 GET /sessions/{sessionId}

- **Roles**: TL/TM of that team, EM.
- **Use**: View session state: current round, timers, members, etc.
- **Response**: session DTO including currentRound, timerRemaining, stats.

6.3 GET /teams/{teamId}/sessions

- List sessions for one team (filters by `status` etc.).

6.4 PATCH /sessions/{sessionId}/control

- **Roles**: TEAM_LEADER (own), EVENT_MANAGER (any) – FR-204, FR-205.
- **Body**: `{ action: "START" | "PAUSE" | "RESUME" | "END" }`.
- **Responsibilities**
  - For `START`:
    - Transition status PENDING → RUNNING.
    - Set `currentRound = 1`.
    - Start timer (5 minutes default—FR-203).
    - Create Round 1 record if not pre-created.
  - For `PAUSE`/`RESUME`:
    - Stop / restart timer service, persist state.
  - For `END`:
    - Finalize current round, close session (status COMPLETED).
    - Ensure all ideas saved (FR-210).

- **Response**: `200` session state.

6.5 POST /sessions/{sessionId}/rounds/{roundNumber}/advance

- **Roles**: SYSTEM (timer), TEAM_LEADER, EVENT_MANAGER.
- **Use**: Explicit transition when timer expires → next round (FR-207, FR-208).
- **Responsibilities**
  - Lock submissions for the round.
  - Perform idea "passing" mapping from each participant's ideas to next participant.
  - Persist new Round record with passed-in ideas visible to each user.
  - Update `currentRound` and timers.
- **Response**
  - `200` with `{ currentRound, previousRoundStatus, passedIdeaMap }`.

6.6 GET /sessions/{sessionId}/rounds

- List all rounds and their status (useful for EM dashboards).

6.7 GET /sessions/{sessionId}/rounds/{roundNumber}

- Detail of a single round: timer state, members that have submitted.

---

7. Ideas (submission & viewing)

Implements FR-201–210, FR-306–307, Scenarios 4,5,8, and "Participate in Session" sequence diagram.

7.1 GET /sessions/{sessionId}/rounds/{roundNumber}/ideas

- **Roles**: TEAM_MEMBER & TEAM_LEADER in that team, EVENT_MANAGER (for oversight).
- **Use**:

    - TMs: See “ideas from previous teammate” and their own ones.
    - TL/EM: Monitor progress. (Scenario 8.)
- **Responsibilities**
    - Enforce idea visibility restriction to team (FR-209).
    - For TMs, show:
        - `previousTeammateIdeas` (3 ideas passed to them),
        - their own previously submitted ideas (if any),
        - status of others as aggregated counts (not full content if you want).

Response:

```
{
 "roundNumber": 2,
 "previousTeammateIdeas": [ { "id": "...", "text": "..." }, ... ],
 "yourIdeas": [ ... ],
 "submissionStatus": { "submittedCount": 4, "totalMembers": 6 }
}
```

-

7.2 POST /sessions/{sessionId}/rounds/{roundNumber}/ideas

- **Roles**: TEAM_MEMBER (and Team Leader if also a member).
- **Use**: Submit the 3 ideas for current round (FR-201, FR-202, FR-306).

Body:

```
{
 "ideas": [
   { "text": "..." },
   { "text": "..." },
   { "text": "..." }
 ]
}
```

-
- **Backend responsibilities**

    - Check:
        - user belongs to team + session is RUNNING + correct roundNumber.
        - not already submitted, or handle update policy.
        - exactly 3 non-duplicate ideas (FR-201).
    - Persist ideas; append to SessionLog; trigger real-time update events (FR-500–501).
- **Response**
    - `201` with created idea IDs and new submission status.

7.3 PATCH /ideas/{ideaId}

- **Roles**: TEAM_MEMBER (own idea) before round lock, TEAM_LEADER / EM maybe for moderation.
- **Use**: Allow minor edits; log changes (audit trail, FR-602).
- **Body**: `{ text }`.
- **Response**: Updated idea.

7.4 GET /sessions/{sessionId}/ideas

- **Roles**: TEAM_LEADER (own team), EVENT_MANAGER.
- **Use**: View aggregated ideas across all rounds (FR-301, FR-305, FR-600).
- **Response**: ideas grouped by round & participant.

---

8. AI Integration (ChatGPT)

Implements FR-400–403, scenarios 6 & 7, AI sequence + activity diagrams.

8.1 POST /ai/sessions/{sessionId}/suggestions

- **Roles**: TEAM_LEADER, EVENT_MANAGER.
- **Use**: Request AI ideas given current topic + ideas (FR-400).
- **Body** (optional):
    - `roundNumber?`,

    - `promptOverride?` (extra instructions).
- **Backend responsibilities**
    - Fetch session, topic, and optionally ideas up to specified round.
    - Build ChatGPT prompt.
    - Call OpenAI API with rate-limit handling and error fallbacks (NFR-202, FR-402).
    - Save AI suggestions as `AIArtifact` linked to session/round.
- **Response**
    - `200 { suggestions: [ { id, text } ] }` or `503` with error message when AI unavailable (Handle AI Error diagram).

8.2 POST /ai/sessions/{sessionId}/summary

- **Roles**: EVENT_MANAGER, TEAM_LEADER (FR-401–402).
- **Use**: Summarize all ideas of a completed session (Scenario 7).
- **Body**: optional config: `{ style, length, language }`.
- **Responsibilities**
    - Validate session COMPLETED.
    - Fetch all ideas and send to ChatGPT.
    - Store structured summary (`AIArtifact` type SUMMARY).
- **Response**
    - `200 { summaryId, summaryText, keyThemes[], notableIdeas[] }`.

---

9. Real-Time & Notifications

REST here is mostly for configuration; actual push is WebSocket/SignalR/FCM as required by non-functional requirements.

9.1 GET /realtime/token

- **Roles**: Authenticated.

- **Use**: Get signed token / channel info to connect to WebSocket (FR-500–501).
- **Response**
  - `{ socketUrl, authToken, channels: ["session-<id>", "team-<id>"] }`.

On the real-time channel, you would push events like:

- `ROUND_STARTED`, `ROUND_PAUSED`, `ROUND_ENDED`
- `IDEA_SUBMITTED`
- `SESSION_STATUS_CHANGED`
- `AI_SUGGESTION_READY`
- `TIMER_TICK` (optional for live countdown)

9.2 POST /notifications/test or /notifications/register-device

- To register mobile push tokens (FR-503).

---

10. Reporting, Export & Audit

Implements FR-300–303, FR-600–602, exportSessionReport scenario, audit requirements.

10.1 GET /reports/events/{eventId}

- **Roles**: EVENT_MANAGER.
- **Use**: High-level dashboard: number of sessions, teams, ideas, AI summaries.
- **Response**
  - Aggregated stats, list of sessions with statuses.

10.2 GET /reports/sessions/{sessionId}

- **Roles**: EM, TL (for own team).
- **Use**: Detailed view of a session (FR-301, FR-305, FR-600).

Response

```
{
 "session": { ... },
 "ideasByRound": { "1": [...], "2": [...] },
 "aiSummaries": [...],
 "aiSuggestions": [...],
 "metrics": {
  "totalIdeas": 90,
  "membersParticipated": 6
 }
}
```

-

10.3 GET /reports/sessions/{sessionId}/export

- **Roles**: EVENT_MANAGER (FR-303, FR-601).
- **Query**: `format=pdf|csv`.
- **Backend responsibilities**
  - Call internal export service (as in Export Session Report sequence diagram).
  - Stream file to client.
- **Response**
  - `200` file download (application/pdf or text/csv).

10.4 GET /audit/sessions/{sessionId}/logs

- **Roles**: EVENT_MANAGER.
- **Use**: View SessionLogs (FR-602, NFR-400).
- **Response**
  - List of events: `{ timestamp, actorId, actionType, payloadSnippet }`.

---

11. Security & Authorization

Rather than separate endpoints, these are cross-cutting rules that you can encode in middleware:

- Before any **management** endpoint (topics, teams, participants, sessions control), verify user role matches allowed roles (FR-700–702).
- On all **session/idea** endpoints, check that:
  - user is a member of the team associated with the session, or
  - user is Event Manager for that event.

---

12. How to use this for task distribution

You can now split backend work roughly as:

1. **Auth & User Service**
   - `/auth/*`, `/participants/*`.
2. **Directory Service**
   - `/events/*`, `/topics/*`, `/teams/*`.
3. **Session Engine**
   - `/sessions/*`, `/sessions/*/control`, `/sessions/*/rounds/*/advance`.
4. **Idea Service**
   - `/sessions/*/rounds/*/ideas`, `/ideas/*`.
5. **AI Service**
   - `/ai/*` + integration with OpenAI.
6. **Realtime & Notifications**
   - `/realtime/token`, `/notifications/*` and WebSocket layer.
7. **Reporting & Audit**
   - `/reports/*`, `/audit/*`.

Each micro-team (or pair) can own one module, following the system controllers already sketched in the sequence diagrams (BrainstormingController,

TeamController, SessionController, AIController, ReportController, AuthController).

If you want, in a next step I can turn this into a formal API spec (OpenAPI/Swagger-style) or help you map specific endpoints to your class diagram (e.g., which controller/class handles which route).