Registration

Zoe, Irene, Abeni

- What route should it have and what should it be scoped under?
  User creation [`POST /api/v1/auth/register`]
  Session creation [`POST /api/v1/auth/login`]
  Authentication [`/api/v1/auth`]
  User profile details [`/api/v1/users`]

- Roles that will be able to access it
  - Guest / Anonymous (unauthenticated visitor) : can register and log in (only from the app)
  - Hacker (participant) : can view and update their own profile
  - Judge / Mentor (evaluator / helper) : can view own profile, and list public user profiles(name, role, public (gitHub) links, team ID)
  - Organizer / Administrator : Full access to all users and status approval

- The HTTP **requests** you can send to it and their content
  - Use this resource to learn more about HTTP requests: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods
- What the **API** should do with that request
- The status code and **response** that is returned to the caller depending on situation
  - Use this resource to learn more about HTTP status codes: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status

  (I put them all together, feel free to add more and correct me if I got something wrong)

1) **Register New User**
   Access: Open to Guest but only from the app. Should be rate limited
   => [`POST /api/v1/auth/register`]
   Email
   Name
   [some sort of OIDC token?]
   Request_role
   API : Validates input, check if email exists, hashes the entered password, create user record, initializes profile record
   => [`201 Created`]successfully registered
   [`400 Bad Request`]invalid email / OIDC error
   [`409 Conflict`]email already registered
2) **Authenticate / Log In**
   Access: Open to Guest but only from the app. Should be rate limited
   => [`POST /api/v1/auth/login`]
   Email
   [some sort of OIDC token?]

API : Look into the database for user by email, use OIDC to verify
=> [200 OK]login success
[401 Unauthorized]invalid email or OIDC token

3) **Fetch Current User Profile**
Access: Open to authenticated users (Hacker, Judge, Organizer, Mentor)
=> [GET /api/v1/users/{user_id}]
API : Read token from HTTP authentication header => fetch users ID, profile & application status from the database
=> [200 OK]profile retrieve success
[401 Unauthorized]missing or expired token
[404 Not Found]user_id DNE

4) **Submit / Update Application Profile**
Access: Open to Hacker
=> [PUT /api/v1/users/{user_id}/application]
*validation
Data - JSONB (some way to mark public/private fields)
School
Major
Graduation_year
T-shirt size
Github_url
Linkedin_url
Resume_url
Public?
API : saves info (T-shirt size, gitHub link, Resume link…) in data
=> [200 OK]application detail updated
[400 Bad Request]validation failed (invalid links)
[401 Unauthorized]unauthenticated

5) **List all applications**
Access: Organizers and registered hackers
=> [GET /api/v1/users]
API :
For organizers: Retrieves list of applicants (complete profile + private application data) with filter options like their application status, check-in status, and t-shirt size…
For registered hackers: retrieves list of public user profile
=> [200 OK]return the list successfully
[403 Forbidden]failed due to insufficient permissions

6) **Review Application Status**
Access: Restricted to Organizers ONLY, and registered hackers to their own status
=> [PATCH /api/v1/users/{user_id}/status]

API : Updates applicants' status [Accepted / Pending / Rejected => remove / Waitlisted] => triggers confirmation email

=> `[200 OK]` success

`[403 Forbidden]` non-organizer attempt

`[404 Not Found]` user_id DNE

- What other parts of the architecture are involved with this
  - Frontend User UI
  - **Authentication Middleware : token verification, enforces role based access control**
  - Database and Cloud Storage (resume files…)
  - Notification(Email / Text)
  - Team management : checks acceptance status before able to form/join teams

Team Creation

Zoe, Irene, Abeni

- What route should it have and what should it be scoped under?
  Team creation [`POST /api/v1/teams`]
  Join team with unique code [`POST /api/v1/teams/join`]

- Roles that will be able to access it
  - Guest / Anonymous (unauthenticated visitor) : no access
  - Pending / Unaccepted Hacker : [`403 Forbidden`]
  - Accepted Hacker (participant) : can create team, join team via code, view team details, leave team, (leader) manage team
  - Judge / Mentor (evaluator / helper) : can view public team profiles and member lists
  - Organizer / Administrator : Full read/write access -create, force add/remove members, dismiss team

- The HTTP **requests** you can send to it and their content
  - Use this resource to learn more about HTTP requests: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods
- What the **API** should do with that request
- The status code and **response** that is returned to the caller depending on situation
  - Use this resource to learn more about HTTP status codes: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status

(I put them all together, feel free to add more and correct me if I got something wrong)

1) **Create a Team**
   Access: Open to Accepted Hackers(No team yet)
   => `[POST /api/v1/teams]`
   `Team_name`
   API : check if the hacker is accepted and not in a team yet, create team record, generates invite code, set maximum team size, assign LEADER
   => `[201 Created]`team created
   `[400 Bad Request]`something missing
   `[403 Forbidden]`trying to add someone not accepted yet to team
   `[409 Conflict]`adding someone who is already in another team
2) **Request to join team**
   Access: Open to Accepted Hackers
   => `[POST /api/v1/teams/{team_id}/{user_id}]`
   `user_id`
   `team_id`
   API : validates user_id, check if target team at maximum capacity, if not then add user to pending members
   => `[200 OK]`successfully joined pending members

[400 Bad Request]team capacity at max
[404 Not Found]target team DNE
[409 Conflict]user already applied to a team

3) **Approve hacker to team**
Access: Open to team leaders (accepted hackers)
=> [POST /api/v1/teams/{team_id}/{user_id}/accept]
user_id
team_id
API : validates user_id as an actual pending member, check if target team at maximum capacity, if not then add user to members
=> [200 OK]successfully joined
[400 Bad Request]team capacity at max | user_id not a pending member
[404 Not Found]target team DNE
[409 Conflict]user already on a team

4) **Fetch Current Team Profile**
Access: Open to Authenticated users (Accepted Hackers / Judge, Mentor, Organizer)
=> [POST /api/v1/teams/{team_id}]
API : fetch team info from team ID, and members' public profile. Associate project submission status.
=> [200 OK]successfully retrieved
[404 Not Found]team DNE

5) **Leave from team / Remove member from team**
Access: Accepted Hackers (Team member self-removal OR Team Leader removing member) and Organizers
=> [DELETE /api/v1/teams/{team_id}/members/{user_id}]
API: Removes users from the team. If LEADER leave then transfer LEADER to another MEMBER, or dismiss team if team EMPTY
=> [200 OK]successful remove or leave
[403 Forbidden]MEMBER trying to remove other MEMBER
[404 Not Found]target MEMBER DNE

6) **[Team Matching???] Search for Team / List of team searching for members**
Access: Open to Authenticated users (Accepted Hackers / Judge, Mentor, Organizer)
=> [GET /api/v1/teams]
Pending members
API: Returns a list of teams seeking members
=> [200 OK]return list of open teams

- What other parts of the architecture are involved with this
  - User Registration Module: STATUS = Accepted

- Database: Team_LEADER and Team_MEMBER role assignments
- Submission Module
- Communication: someone joins team notification?

@Evan and other events people - we are wondering what y'all would like for team matching in terrier? namely,

- are we trying to do team matching in the app, and if so, to what extent?
- do we want to designate a leader for each team?
- do we want hackers to be able to request to join a team, and then they will be approved? and/or do we want hackers to be able to be invited, and then the hacker will accept?
- should hackers be able to apply to multiple teams? like... collect a shopping cart full of teams?