Report

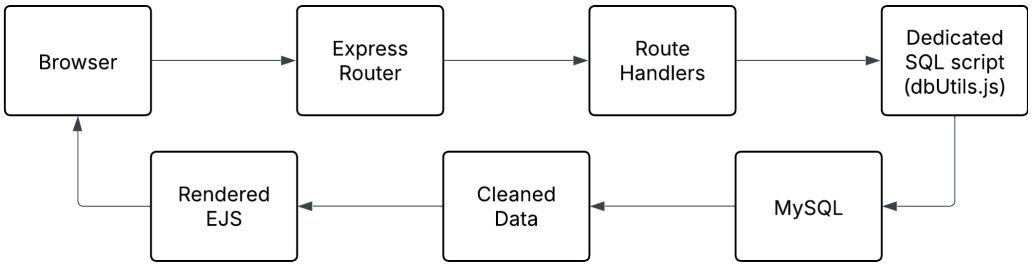
Outline

Runnr is a community-driven fitness tracking application that allows users to log runs, set goals, follow other runners, and stay motivated through an activity feed. The app helps users improve their health by combining fitness tracking with social interaction. It is built using Node.js, Express, EJS, and MySQL. Users can create accounts, update profiles, join goals, log their progress, and search for other users or goals. Persistent data is stored in a MySQL database. The application is deployed on the Goldsmiths virtual server and is fully installable from the provided GitHub repository.

Architecture

The application uses a two-tier structure.

- Application Tier - Node.js + Express for routing, controllers, middleware, session handling, authentication, and rendering EJS templates.
- Data Tier - MySQL stores all persistent data (users, runs, goals, follows). Database access is encapsulated in reusable utility functions.

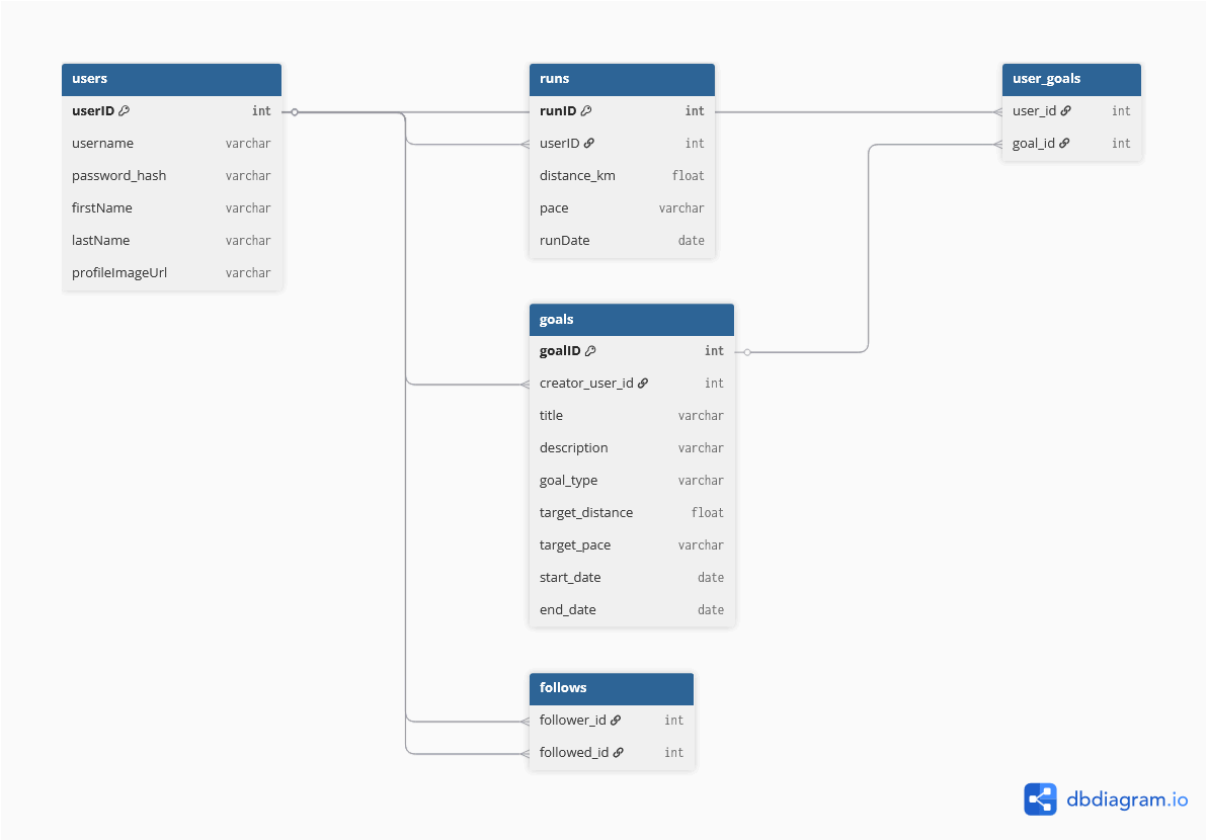


Data Model

The system uses a relational schema representing users, runs, goals, and social interactions.

Tables

- **users** – stores account info, profile data, password hash.
- **runs** – individual user-submitted run entries (distance, pace, date).
- **goals** – goals created by users with target pace/distance and dates.
- **user_goals** – join table mapping users to the goals they joined.
- **follows** – records user → user follow relationships.



User Functionality

Runnr provides a clean, user-friendly interface with multiple interconnected features:

Home Page

A dashboard showing recent activity of other users the logged in user follows, and also some suggested users to follow and goals to join

Authentication

Users can register and log in (default credentials provided: *gold / smiths*). Password validation matches coursework rules and is simply a minimum of 6 characters. Sessions persist login state server-side.

Profiles

Each user has a public profile displaying:

- Name, username, avatar
- Follow/unfollow button
- Stats for followers, following, runs, goals
- A user-specific activity feed

Goals

Users can:

- Create new goals (form submission → MySQL insertion)
- Join goals
- View goal details (title, description, target pace/distance, members, activity)

Runs

Users can:

- Log new runs
- View run details
- Scroll through personal and global run lists

Search

A search page lets users find:

- Other users
- Goals

Advanced Techniques

User uploaded profile pictures

The application allows users to upload their own profile images. When a user selects a new picture, the server saves the file and stores its filename in the database so the correct image appears on their profile.

```
// UPDATE PROFILE PICTURE
router.post(
  '/profile-picture',
  redirectLogin,
  profilePics.uploadProfilePic,
  async (req, res, next) => {
    try {
      const userID = req.session.loggedUser.userID;
      const oldUrl = req.session.loggedUser.profileImageUrl;

      if (!req.file) {
        return res.status(400).send('No file uploaded.');
      }

      const newUrl = profilePics.getProfileImageUrlFromFile(req.file);
```

```
      await dbUtils.updateUserSetting('profile_image_url', newUrl,
userID);
      req.session.loggedUser.profileImageUrl = newUrl;

      if (oldUrl && oldUrl !== newUrl) {
        profilePics.deleteProfileImageByUrl(oldUrl);
      }

      res.render('settings.ejs', {
        errorsToDisplay: [],
        successMessagesToDisplay: [
          messages.AUTH.UPDATE.PROFILE_PICTURE_UPDATED_SUCCESSFULLY,
        ],
      });
    } catch (err) {
      console.error(err);
      next(err);
    }
  },
);
```

Found in routes/settings.js

EJS Layout and partials

The project uses a layout file and many partials to keep the interface consistent and avoid repetition. The main layout (`layouts/main.ejs`) contains shared page elements, and pages insert their own content into this structure.

Reusable components such as `profileCard.ejs`, `activityCard.ejs`, `runsList.ejs`, and `navbar.ejs` allow complex UI elements to be generated with dynamic data.

```
<%- include("partials/profileCard.ejs", {
        user: user,
        followers: [],
        following: [],
        loggedUser: loggedUser,
        isOwner: false,
        isFollowing: false,
        displayMode: "summary",
        actionMessage: null
```

```
    }) %>
```

Found in views/searchResults.ejs

Scheduled cron jobs for checking goal completion

The application uses a scheduled cron job to automatically update goals in the background. Instead of waiting for a user to trigger an update, the server periodically checks for goals whose end dates have passed and updates their status accordingly.

```
const cron = require('node-cron');

const goalUtils = require('../utils/goalUtils');

cron.schedule('0 3 * * *', async () => {
  console.log('[CRON] Checking goal statuses...');

  try {
    await goalUtils.checkAllGoals();
    console.log('[CRON] Done.');
  } catch (err) {
    console.error('[CRON ERROR]', err);
  }
});
```

Found in cron/goalCron.js

AI Declaration

AI tools were used to assist with debugging queries, assisting with ideas for styling/design, as well as generating a bunch of test data to populate the MySQL database. All the final code was written by myself.

Extra note

A small error in the button link has rendered the settings page inaccessible from the navbar. To use it you have to be logged in "https://www.doc.gold.ac.uk/usr/224/settings" directly from the address bar.