Assignment 1 Report | QR Forge

Design, Implementation, and Reflection of a Web Application for QR Code Generation

https://github.com/JSebastianIEU/Assignment_1_qr_forge_Devops.git

Student: Juan Sebastián Peña Donneys

Professor: Borja Serra Planelles

Course: DevOps and Software Engineering

IE University - 2025

Introduction and Business Problem

In today's digital ecosystem, QR codes have become indispensable. Businesses rely on them to share menus, promotions, and resources instantly; universities use them to distribute course materials; and individuals increasingly adopt them for networking or portfolio sharing. Despite their widespread use, many available tools to generate QR codes are limited or lack proper management features.

The business problem QR Forge aims to solve is the lack of a free, simple, and extensible QR management system. Users often need not only to create QR codes, but also to customize their appearance, save them for later, and manage their collection efficiently. QR Forge proposes a lightweight web app that combines these elements into a minimal yet complete solution.

This project also served as an academic exercise in software engineering practices: it required identifying a problem, designing diagrams and models, following an SDLC, and building iteratively. The reflection on how such a system could scale under DevOps contexts adds further value.

Software Development Life Cycle (SDLC) and Chosen Model

Before beginning development, I invested significant time in planning. I produced the following key documents, now included in the annex:

- **Application Flow Diagram (PDF)** – mapping the entire user journey from Home to Generator, Login, Signup, Profile, and History.
- **Application Data Model (ERD) (PDF)** – defining the relational database structure with users and QR items.
- **Application Architecture Sequence Diagram (PDF)** – showing how requests flow through routers, services, and the database. julian franco (no mandar tweets, reparacion) todos son iguales JAC, cerebros fugados (atencion psicologica) (fronteras invisibles, casas de sicariato, gota a gotas) (daniel)
- Not created before, but in the annex, you can also find:
    - Application API Documentation (PDF) – automatically generated from FastAPI's Swagger, listing endpoints and models.
    - Prompts Used (PDF) – documenting every AI interaction that supported implementation.

For the SDLC, I followed an Agile-inspired iterative approach. The project was broken into sprints, each producing a working increment: authentication, generator, history, and

profile. This allowed flexibility to integrate changes (e.g., making QR titles optional, refining preview-before-save logic).

The Agile model ensured that after each sprint, the application remained usable, aligning well with the assignment's emphasis on multiple commits across time.

Application Architecture

The QR Forge system follows a three-layer architecture:

1. Frontend – Built with HTML, CSS, and JavaScript, it provides a clean and consistent UI across pages. The navigation sidebar with SVG icons ensures usability. The design is minimal, in line with the early wireframes.
2. Backend – Powered by FastAPI, chosen for its speed and built-in OpenAPI support. Routes are separated into dedicated modules:
    - auth – for signup, login, and logout.
    - user – for profile operations.
    - qr – for QR creation, customization, history, and deletion.
    - export – for file downloads (SVG/PNG).

This modularity matches the Application Architecture Sequence Diagram (PDF), ensuring clarity between controller, service, and database layers.

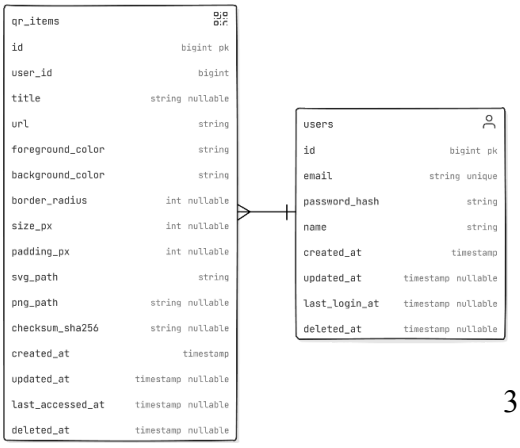
3. Database – A SQLite relational schema defined in models.py. It directly follows the Application Data Model (ERD):
    - users with fields for email, hashed password, and timestamps.
    - qr_items with URL, title, customization metadata, and timestamps. Indexes on email and user_id improve efficiency for login and history queries.

This architecture is modular and future-proof, allowing future scaling to Docker and cloud environments.

Data Model

The Application Data Model (ERD PDF) forms the backbone of persistence. Two core entities were implemented:

- Users – Each has a unique email, a hashed_password, and timestamps (created_at, last_login_at). Integrity is



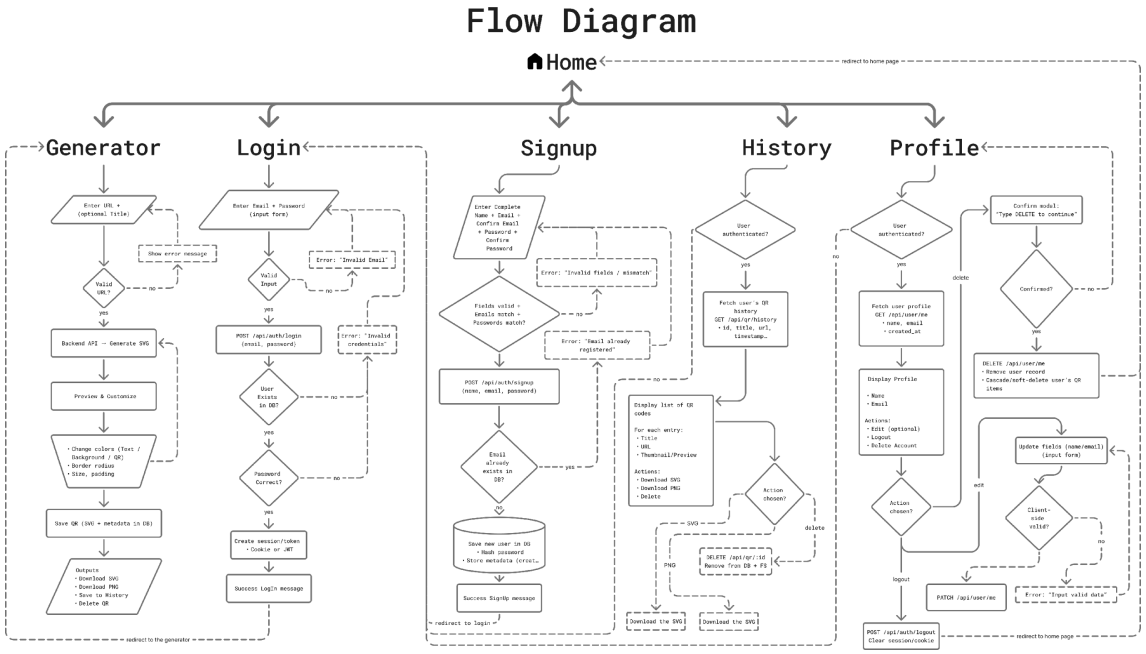
enforced through uniqueness constraints and foreign key relations.

- QR Items – Each entry is tied to a user. It stores metadata including title, url, foreground_color, background_color, and paths to the SVG files. The addition of updated_at helps track modifications.

Defaults (e.g., black foreground, white background) simplify inserts, while Pydantic validation enforces URL correctness. Indexes guarantee that even with many QRs, history remains fast to load. The alignment between the ERD and the implemented schema shows how early planning simplified the coding process.

Flow and User Journeys

The Application Flow Diagram (PDF) illustrates the system's logic:



- Authentication – Only authenticated users can access Profile and History.
- Generator – Users input a URL and optional title, preview the QR, customize colors, and then choose to save. This ensures only finalized QRs are persisted.
- History – A grid view shows all saved QRs with options to download (SVG/PNG) or delete. Updates are reflected instantly in the frontend.
- Profile – Displays account information, logout option, and account deletion.

The implemented app follows these flows precisely, showing strong alignment between design and code.

Design Reflection

One of the most valuable aspects of this project was the commitment to a **design-first approach**. Instead of rushing into coding, I first developed a series of diagrams and models that would serve as a roadmap for implementation. This methodology allowed me to minimize uncertainty, avoid redundant work, and maintain a clear sense of direction throughout the sprints.

The **flow diagram**, for example, was not just a theoretical artifact. It actively shaped the way user journeys were implemented, ensuring that every major action—authentication, QR generation, preview-before-save, and history management—was consistent with the original design. Similarly, the **entity–relationship diagram (ERD)** guided the construction of the database schema. By clearly defining entities, relationships, and constraints before writing SQL or models in FastAPI, I avoided later conflicts and established a solid data layer from the start.

The **sequence diagram** contributed by clarifying the modular architecture of the application. It helped me separate responsibilities into routers, services, and database layers, reinforcing the idea that good design reduces complexity in implementation. The **API documentation**, automatically produced by FastAPI and exported as a PDF, became a mirror of the diagrams: it confirmed that every endpoint defined during design existed in practice and was testable. Finally, the **Prompts Used (Quino)** annex provides an additional layer of accountability by documenting the role of AI assistance in generating and refining code across sprints.

Scalability and DevOps Adaptation

Although QR Forge was developed to run locally, its modular architecture was designed with future scaling in mind. Migrating from SQLite to a production database like PostgreSQL or MySQL would enable higher concurrency, while containerization with Docker would simplify deployment across environments.

Static assets such as QR files could be offloaded to cloud storage (e.g., AWS S3), and CI/CD pipelines would automate testing and releases. Security could also be reinforced through more advanced authentication methods like JWT or OAuth.

The **Application Architecture Sequence Diagram (PDF)** already illustrates this layered structure, showing how the system could transition smoothly into a cloud-native environment. Even as an academic prototype, QR Forge provides a realistic foundation for DevOps practices.

AI Usage

AI was used as a development companion, but not as an automated code generator. Two tools played different roles:

- Codex – Used primarily for sprint generation: backend routes, schemas, and database logic. It produced detailed code for each sprint milestone, accelerating repetitive tasks.
- GitHub Copilot – Used as a debugging and refinement assistant. While Codex generated structure, Copilot fixed syntax errors and suggested small improvements.

To ensure transparency, all prompts are documented in the Prompts Used (PDF) annex. Alongside AI, I reinforced my learning by consulting tutorials and documentation:

- FastAPI Tutorial – Python Web APIs
- QR Code Generator with Python
- Build and Customize QR Codes in Python

These tutorials helped me understand how FastAPI structures projects, how Python QR libraries work, and how to integrate backend and frontend flows. Combining this external knowledge with the diagrams ensured I had a clear structure to provide to Codex. In turn, this maximized Codex's usefulness, while Copilot's debugging closed gaps Codex left. This balanced approach shows not blind dependence on AI, but intelligent integration of human planning, external knowledge, and AI assistance.

Conclusion

QR Forge demonstrates the value of structured software development. By identifying the business problem, carefully planning with diagrams, following an Agile SDLC, and responsibly integrating AI tools, I was able to deliver a complete web application within the scope of the assignment.

The app allows users to generate, customize, save, and manage QR codes, with authentication and history support. The annexes Application Flow Diagram, Application Data Model (ERD), Application Architecture Sequence Diagram, Application API Documentation, and Prompts Used provide the detailed technical foundation and evidence of planning. This combination of manual design, iterative coding, and AI-assisted refinement resulted in a system that is functional today and scalable tomorrow.

Notes

In addition to manual validation, a suite of automated tests was written using **pytest** and FastAPI's **TestClient.** In total, **12 tests** cover the authentication lifecycle (signup, login, logout, invalid credentials), QR generation (preview, create, download, delete), history access, and profile updates. These tests confirm that authentication guards are enforced, input validation works as intended, and user-specific data remains isolated.

ANEX

AI Usage | Prompts and Outputs

ChatGPT(OpenAI)

During the early stages of developing QR Forge, ChatGPT served as a foundational design and ideation partner. It was used primarily to expand preliminary concepts, provide structural guidance, and help establish a coherent architectural vision for the application. This collaboration reflected an intentional design-first mindset, where every technical and creative decision was grounded in planning before implementation.

The interaction process with ChatGPT was not about letting the model "create the project," but rather using it as a cognitive amplifier—a way to extend reasoning, explore possibilities, and transform abstract ideas into structured blueprints for development.

1. Ideation and Concept Expansion

   The project originated from two ideas I conceived independently:

   - A web application capable of generating customizable QR codes from user-provided links, offering SVG and PNG export options.
   - A membership card system that would allow users to generate verifiable digital credentials including a photo and personal information.

   At this stage, I turned to ChatGPT to help expand on these ideas.

   My prompt asked the model to elaborate their functionality, potential architecture, and scalability paths. The intention was to understand how each concept could evolve into a full web application aligned with DevOps and cloud-based deployment practices.

   ChatGPT responded with detailed functional descriptions, emphasizing modular design, API-driven architecture, and extensibility through database integration.

   This exchange helped me visualize both projects from a higher technical perspective and make an informed decision to proceed with the QR generator due to its practicality and clearer technical scope.

   The prompting strategy here demonstrated structured ideation: I supplied constraints (e.g., use FastAPI, focus on data persistence, consider scalability) while leaving enough creative room for the model to propose implementation strategies.

2. Project Initialization and Technical Setup

Once the main concept was defined, I used ChatGPT as a guide to organize the project's initial structure. I crafted prompts requesting help with environment setup, project scaffolding, and repository preparation.

For example, I asked:

"Help me prepare my FastAPI project structure and decide how to organize folders for routes, models, services, and database."

The responses led to the creation of a well-organized modular system that divided the application into distinct layers:

/routers for request handling,

/services for business logic,

/models for SQLModel database entities, and

/schemas for validation and serialization.

ChatGPT's guidance in this phase reflected a mentor-like role—it ensured I followed proper software engineering patterns while maintaining flexibility for future sprints. This phase also set the foundation for integrating testing frameworks and database management tools, though the later implementations were handled manually.

3. Interface Design and Mockup Collaboration

After the backend structure was set, I moved to the visual design stage. I prompted ChatGPT with requests like:

"Help me design a minimal, responsive QR generator interface using HTML, CSS, and JavaScript."

"Center the generator box, add a clean layout, and make the generated QRs display in a dynamic grid."

This iterative process was central to refining the UI experience. ChatGPT provided several foundational templates—clean layouts, centered forms, and hover animations—which I later adapted manually in Figma to better align with the desired aesthetic.

The model's feedback loop allowed me to iterate quickly: each prompt represented a micro-experiment in user interaction design. I would describe how I wanted the UI to feel—interactive, consistent, and intuitive—and ChatGPT would provide base HTML/CSS code to be refined further.

The output was not a final design but a structured skeleton, a starting point that helped me accelerate the mockup development in Figma. This early combination of generated code and manual design helped me achieve both speed and creative control.

4. Diagram Generation and Design Refinement

   To ensure every architectural decision was well-documented, I used ChatGPT to help build the first versions of my system diagrams.

   The prompts included instructions such as:

   "Generate a basic flow diagram for the QR Forge app including authentication, profile, and QR generation paths."

   "Create an ERD showing users and QR items with their attributes and relationships."

   ChatGPT generated the base logical structures, which I later manually redesigned and styled.

   The Entity Relationship Diagram (ERD) was initially created with ChatGPT's SQLModel mapping guidance, then built in MySQL Workbench, and finally refined visually using Eraser.io and Figma to achieve a cleaner and more professional look.

   Similarly, the Application Flow Diagram and Architecture Sequence Diagram originated from basic ChatGPT sketches but were later redesigned from scratch in Figma.

   These diagrams became key visual anchors for planning each sprint and understanding system dependencies.

   The final outputs attached as Application Flow Diagram.pdf, Application Architecture Sequence Diagram.pdf, and Application Data Model (ERD).pdf reflect the combination of AI-generated structure and human-driven refinement.

5. First Iteration and Code Generation

   The first version of the QR Forge codebase was also developed with ChatGPT's assistance.

   I prompted the model for practical implementation details—how to create API endpoints for user authentication, QR creation, and history management using FastAPI and SQLModel.

ChatGPT generated initial versions of routes, schemas, and database interactions.

This version was not final, but it established a solid technical baseline that guided the first working prototype. From there, I iteratively improved the code, debugging and extending it manually based on deeper architectural understanding and testing. This approach reflected a structured co-development model:

- ChatGPT handled boilerplate and structure, I handled validation, logic, and refinement.

It allowed me to focus on system consistency, enforcing a modular and testable architecture while accelerating early development through AI-assisted scaffolding.

Codex (OpenAI)

6. Transition to Codex: code-oriented iteration driven by a pre-engineered prompt

   After I completed the first design-centric iteration with ChatGPT (ideation, initial structure, and a working baseline), I shifted to Codex for code-focused development. Crucially, I did not send Codex ad-hoc requests; instead, I crafted a comprehensive, repo-aware "master prompt" with ChatGPT's help. ChatGPT helped me tighten the scope, spell out grading criteria, name concrete endpoints, and sequence the work into small, reviewable commits. Only then did I deliver the final, structured brief to Codex.

   a. The master prompt (engineered with ChatGPT, executed by Codex)

      Before contacting Codex, I asked ChatGPT to help me consolidate everything the model would need to act like a focused engineering collaborator. The refined prompt I then gave Codex (abridged here for narrative clarity) was:

      "Please analyze all these documents and my current codebase. I created diagrams and page mockups that describe how everything should work. Use them to guide the plan and proposed changes.

      Context: QR Forge app (FastAPI backend; HTML/CSS/JS frontend; SQLite/MySQL). Local run only (no DevOps). Repo includes: Flow Diagram, Architecture Sequence Diagram, ERD, PNG mockups, and navbar SVG icons.

      Grading criteria (explicit):

- *Features (60%): generator (+preview/customization), history (list/download/delete), auth (signup/login/logout + profile).*
- *Docs (20%): README with setup/run/screenshots; reflection linked to Flow/Sequence/ERD PDFs.*
- *Code & Git (20%): clean modular code, basic error handling, multiple meaningful commits.*

What I want (local, incremental, agile):

- *Audit repo vs diagrams; list gaps/mismatches.*
- *Map to grading criteria; propose minimal upgrades.*
- *Sprints 1–4 with commit-by-commit plans and diffs/snippets; no big rewrites.*
- *Endpoints & pages checklist, acceptance criteria, and constraints (local run only).*

Start with Sprint 1 when I say 'start'; wait for me between sprints."

I also embedded file-level expectations (e.g., /api/auth/*, /api/user/me, /api/qr/history, /api/qr/{id}/download?format=svg|png, schema alignment with the ERD, icons under /assets/icons/) so Codex would stay grounded in the exact deliverables. That up-front rigor—co-designed with ChatGPT—meant Codex could produce targeted code and plans instead of generic boilerplate.

b. Codex output: repo audit → sprint roadmap

Codex first returned a precise audit of the repository, marking what existed (basic QR CRUD, CSV export, minimal template) and what was missing (auth, per-user history, customization fields, PNG downloads, multi-page UI, navbar icons, README, and Swagger examples). It then mapped those gaps to my rubric and produced a four-sprint plan with commit-by-commit guidance. That plan became the backbone of the implementation phase.

7. Sprint 1 with Codex: data & authentication foundations

Following my "start with sprint 1" instruction, Codex generated concrete changes that aligned the codebase with the ERD and introduced authentication:

- Models & DB. It added a User table and enriched QRItem (timestamps, color defaults, padding, border radius, FK to users, useful indexes), plus an updated engine/init path—exactly what my ERD required.

- Security & auth. It scaffolded password hashing and JWT token utilities, then exposed /api/auth/signup, /api/auth/login, /api/auth/logout, and /api/user/me.
- Ownership. QR routes became user-scoped; history queries filtered by user_id; destructive actions enforced ownership.
- Testing scaffold. It outlined pytest fixtures for an in-memory DB and basic lifecycle tests (signup→login→create→list→download→delete), which I later expanded.

This sprint gave me a secure, ERD-compliant core. Because the plan was pre-engineered, Codex's diffs were incremental and readable, which made the first "green run" straightforward.

8. Sprint 2 with Codex: generator customization and real assets

In Sprint 2 I asked Codex to implement the customization pipeline end-to-end—the point where many student projects remain superficial. Guided by my diagrams and UI mockups (and the prompt's acceptance criteria), Codex:

- Extended the schema so the generator accepts foreground_color, background_color, size, padding, and border_radius (matching the ERD defaults).
- Factored an asset service (services/qr.py) that produces both SVG and PNG from the same rendering function, ensuring preview, downloads, and history are visually consistent.
- Persisted metadata and paths so thumbnails and history cards could reflect the user's customizations.

This sprint turned "a QR exists" into "my custom QR exists and is reproducible," which was central to the Features 60% criterion.

9. Sprint 3 with Codex: multi-page UI & icon-based navigation

My master prompt asked Codex to respect the mockups (Home, Login, Signup, Profile, Generator, History) and wire SVG icons from /assets/icons/. Codex responded by:

- Introducing a shared Jinja layout and dedicated pages for each screen (matching my wireframes).
- Mounting /assets and updating CSS/JS to support the left-rail icon navbar and auth-aware views.

- Rebuilding the front-end controller so generator forms submit the full customization payload, previews update, history reflects saves/deletes, and downloads honor auth.

I did encounter mismatches with my intended UX (e.g., preview behavior, login gating, a few layout details). Here, my follow-up prompts (again refined with ChatGPT) were specific: “the preview should not save until I click Save,” “apply foreground/background to the actual QR, not just a container,” “enforce login before any preview/save,” “remove center text/icon.” Codex iterated quickly against those precise constraints.

a. Sprint 3.5 (targeted fixes prompted by me)

I asked for three focused adjustments:

- Preview workflow. Add /api/qr/preview, re-render on every tweak, and only persist on Save.
- Account controls. Profile page gains Edit and Delete account (with backend cleanup).
- Visual fidelity. Foreground/background, padding, and border radius affect the generated QR itself, thumbnails update immediately, transparent background is supported.

Codex produced the changes, I validated them against the UI and history screens, and the experience finally matched both my flow diagram and the generator mockups.

10. Sprint 4 with Codex: polish, docs pointers, and acceptance checks

For Sprint 4 I kept Codex tightly scoped: strengthen validation, enrich OpenAPI summaries/examples, clarify README (local run, screenshots, links to the Flow, Architecture Sequence, and ERD PDFs), and add a small test expansion. While I ultimately wrote the longer report narrative myself, Codex’s output ensured the /docs surface was coherent and that local-run acceptance criteria were demonstrably met:

- Run locally with uvicorn.
- Swagger lists all routes with examples.
- From the UI: Login/Signup → Profile, Generator → preview/customize → Save → History, download SVG/PNG, delete updates History immediately.
- Navbar icons visible from /assets/icons/.

11. Why this worked: prompt engineering as process control

    The success of the Codex phase wasn't accidental. Three prompt-engineering choices—developed with ChatGPT first—made the difference:

    - Constrained scope + explicit rubric. Codex was told exactly what "full marks" meant (features, docs, code+Git), which focused its suggestions and code.
    - Repo-aware audit + delta plan. I required an initial audit, then sprints with commit-by-commit plans and file paths—this kept changes incremental and reviewable.
    - Acceptance criteria + hold points. I made Codex wait for my go between sprints. When UX differed from my mockups, I issued surgical prompts (e.g., "preview must re-render and not save; enforce login") to steer corrections without undoing previous work.

    In short, ChatGPT shaped the strategy and the prompt; Codex executed the code. This division of labor mirrors real teams: architects articulate standards and acceptance tests; implementers deliver deltas that satisfy those tests.

GithubCopilot

12. Handover to GithubCopilot

    Finally, once Codex had landed the structural increments, I used GitHub Copilot for inline completion and micro-refactors during day-to-day editing (e.g., tightening a fetch handler, tidying CSS, or suggesting a small helper). Copilot was not asked to design or plan; it was the fast lane for finishing touches inside the IDE.

13. Closing note

    The Codex phase demonstrates that, with a well-engineered master prompt (co-designed with ChatGPT), an AI coding model can work as a disciplined teammate: auditing, planning, and shipping incremental, testable changes that align with diagrams, UI mockups, and grading criteria—without rewriting the project from scratch.

Application API Documentation

QR Forge 1.0.0 OAS 3.1

/openapi.json

Generate, preview, customise, and manage QR codes locally with FastAPI.

QR Forge - Website
Send email to QR Forge
MIT

Authorize

auth Authentication endpoints for signing up, logging in, and logging out.

POST /api/auth/signup Create a new user account

Parameters Try it out

No parameters

Request body required application/json

Example Value | Schema

```
{
  "email": "user@example.com",
  "full_name": "Sample User",
  "password": "strongpass123"
}
```

Responses

Code	Description	Links
201	Newly created user profile Media type application/json Controls `Accept` header. **Example Value** | Schema	*No links*

Code	Description	Links


```
{
  "created_at": "2024-01-01T12:00:00Z",
  "email": "user@example.com",
  "full_name": "Sample User",
  "id": 1,
  "updated_at": "2024-01-02T12:00:00Z"
}
```


422 Validation Error — *No links*

Media type: application/json

Example Value | Schema


```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```


POST `/api/auth/login` Authenticate and receive an access token

Parameters — Try it out

No parameters

Request body required — application/json

Example Value | Schema


```
{
  "email": "user@example.com",
  "password": "strongpass123"
}
```


Responses

Code	Description	Links
200	Bearer token for subsequent requests	*No links*

Media type: application/json

Controls `Accept` header.

Example Value | Schema

Code	Description	Links


```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "token_type": "bearer"
}
```


422 Validation Error — *No links*

Media type: application/json

Example Value | Schema


```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```


POST `/api/auth/logout` Client-side logout acknowledgement

Parameters | Try it out

No parameters

Responses

Code	Description	Links

200 Successful Response — *No links*

Media type: application/json

Controls `Accept` header.

Example Value | Schema


```
{
  "additionalProp1": {}
}
```


users

Profile management endpoints for viewing, updating, or deleting the current user.

GET `/api/user/me` Return the authenticated user's profile

Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	Current user record Media type application/json Controls `Accept` header. **Example Value** | Schema `{ "created_at": "2024-01-01T12:00:00Z", "email": "user@example.com", "full_name": "Sample User", "id": 1, "updated_at": "2024-01-02T12:00:00Z" }`	*No links*

DELETE /api/user/me Delete the authenticated user and all owned QR codes

Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	Confirmation payload Media type application/json Controls `Accept` header. **Example Value** | Schema `{ "additionalProp1": {} }`	*No links*

PATCH /api/user/me Update profile details

Parameters | Try it out

No parameters

Request body required | application/json

Example Value | Schema

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

Responses

Code	Description	Links
200	Updated user record Media type: application/json Controls `Accept` header. **Example Value** | Schema `{ "created_at": "2024-01-01T12:00:00Z", "email": "user@example.com", "full_name": "Sample User", "id": 1, "updated_at": "2024-01-02T12:00:00Z" }`	*No links*
422	Validation Error Media type: application/json **Example Value** | Schema `{ "detail": [ { "loc": [ "string", 0 ], "msg": "string", "type": "string" } ] }`	*No links*

qr QR generation, preview, download, and history endpoints scoped to the authenticated user.

POST /api/qr/preview Render a customised QR preview without saving

Parameters Try it out

No parameters

Request body required application/json

Example Value | Schema

```
{
  "background_color": "#ffffff",
  "border_radius": 20,
  "foreground_color": "#1f3a93",
  "padding": 12,
  "size": 320,
  "title": "Campaign QR",
  "url": "https://example.com"
}
```

Responses

Code	Description	Links
200	Inline base64 PNG and SVG markup Media type: application/json Controls `Accept` header. **Example Value** | Schema	*No links*

```
{
  "png_data": "iVBORw0KGgoAAAANSUhEUg...",
  "svg_data": "<svg xmlns=...>"
}
```

Code	Description	Links
422	Validation Error Media type: application/json **Example Value** | Schema	*No links*

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
```

Code	Description	Links

```
    }
  ]
}
```

GET `/api/qr` List QR codes owned by the authenticated user

Parameters Try it out

No parameters

Responses

Code	Description	Links
200	Successful Response Media type: application/json Controls `Accept` header. **Example Value** | Schema	*No links*

```
[
  {
    "id": 0,
    "user_id": 0,
    "title": "string",
    "url": "string",
    "foreground_color": "#000000",
    "background_color": "#FFFFFF",
    "size": 256,
    "padding": 10,
    "border_radius": 0,
    "overlay_text": "stri",
    "svg_path": "string",
    "png_path": "string",
    "created_at": "2025-10-01T09:49:48.413Z",
    "updated_at": "2025-10-01T09:49:48.413Z"
  }
]
```

POST `/api/qr` Persist a customised QR code

Parameters Try it out

No parameters

Request body required — application/json

Example Value | Schema

```
{
  "background_color": "#ffffff",
  "border_radius": 20,
  "foreground_color": "#1f3a93",
  "padding": 12,
  "size": 320,
  "title": "Campaign QR",
  "url": "https://example.com"
}
```


Responses

Code	Description	Links
201	Saved QR item with asset paths Media type application/json Controls `Accept` header. **Example Value** | Schema	*No links*


```
{
  "id": 0,
  "user_id": 0,
  "title": "string",
  "url": "string",
  "foreground_color": "#000000",
  "background_color": "#FFFFFF",
  "size": 256,
  "padding": 10,
  "border_radius": 0,
  "overlay_text": "stri",
  "svg_path": "string",
  "png_path": "string",
  "created_at": "2025-10-01T09:49:48.416Z",
  "updated_at": "2025-10-01T09:49:48.416Z"
}
```


Code	Description	Links
422	Validation Error Media type application/json **Example Value** | Schema	*No links*


```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```


GET /api/qr/history Alias for listing QR history

Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	Successful Response Media type application/json Controls `Accept` header. **Example Value** | Schema	*No links*


```
[
  {
    "id": 0,
    "user_id": 0,
    "title": "string",
    "url": "string",
    "foreground_color": "#000000",
    "background_color": "#FFFFFF",
    "size": 256,
    "padding": 10,
    "border_radius": 0,
    "overlay_text": "stri",
    "svg_path": "string",
    "png_path": "string",
    "created_at": "2025-10-01T09:49:48.418Z",
    "updated_at": "2025-10-01T09:49:48.418Z"
  }
]
```


DELETE /api/qr/{item_id} Delete a saved QR code

Parameters

Try it out

Name	Description
item_id * required `integer` *(path)*	item_id

Responses

Code	Description	Links
200	Confirmation payload Media type application/json	*No links*

Code	Description	Links

Controls `Accept` header.

Example Value | Schema

```
{
  "additionalProp1": {}
}
```

422 Validation Error — *No links*

Media type

application/json

Example Value | Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```

GET /api/qr/{item_id}/download Download a saved QR code

Parameters

Try it out

Name	Description
item_id * required `integer` *(path)*	item_id
format `string` *(query)*	*Default value* : svg svg *pattern: ^(svg|png)$*

Responses

Code	Description	Links
200	Binary SVG or PNG stream	*No links*

Media type

application/json

Controls `Accept` header.

Code	Description	Links

Example Value | Schema

```
"string"
```

422 Validation Error *No links*

Media type

application/json

Example Value | Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```

export CSV export of the authenticated user's QR history.

GET `/api/export/csv` Export the authenticated user's QR history as CSV

Parameters Try it out

No parameters

Responses

Code	Description	Links

200 CSV stream containing saved QR metadata *No links*

Media type

application/json

Controls `Accept` header.

Example Value | Schema

```
"string"
```

default

GET / Serve the landing page

Parameters | Try it out

No parameters

Responses

Code	Description	Links
200	Successful Response Media type: text/html Controls `Accept` header. **Example Value** | Schema `string`	*No links*

GET /generator Serve the QR generator UI

Parameters | Try it out

No parameters

Responses

Code	Description	Links
200	Successful Response Media type: text/html Controls `Accept` header. **Example Value** | Schema `string`	*No links*

Code	Description	Links

GET /history Serve the saved history UI

Parameters Try it out

No parameters

Responses

Code	Description	Links
200	Successful Response Media type text/html Controls `Accept` header. **Example Value** | Schema `string`	*No links*

GET /profile Serve the profile management UI

Parameters Try it out

No parameters

Responses

Code	Description	Links
200	Successful Response Media type text/html Controls `Accept` header. **Example Value** | Schema `string`	*No links*

Code	Description	Links

GET `/login` Serve the login UI

Parameters Try it out

No parameters

Responses

Code	Description	Links
200	Successful Response Media type text/html Controls `Accept` header. **Example Value** | Schema `string`	*No links*

GET `/signup` Serve the signup UI

Parameters Try it out

No parameters

Responses

Code	Description	Links
200	Successful Response Media type text/html Controls `Accept` header. **Example Value** | Schema `string`	*No links*

Code	Description	Links

GET `/health` Simple health check

Parameters Try it out

No parameters

Responses

Code	Description	Links
200	Successful Response Media type: application/json Controls `Accept` header. **Example Value** | Schema `{ "additionalProp1": {} }`	*No links*

Schemas

HTTPValidationError

Collapse all `object`

- `detail` > Expand all `array<object>`

QRCreate

Collapse all `object`

- `title*` `string`
- `url*` `string` `uri` `[1, 2083] characters`
- `foreground_color` > Expand all `string` `matches ^#[0-9a-fA-F]{6}$`
- `background_color` > Expand all `string` `matches ^(#[0-9a-fA-F]{6}|transparent)$`
- `size` > Expand all `integer` `[128, 1024]`
- `padding` > Expand all `integer` `[0, 128]`
- `border_radius` > Expand all `integer` `[0, 120]`
- *Example* > Expand all `object`

QRItem ^ Collapse all object

- id > Expand all (integer | null)
- user_id* integer
- title > Expand all (string | null)
- url* string
- foreground_color > Expand all string ≤ 7 characters
- background_color > Expand all string ≤ 7 characters
- size > Expand all integer [64, 1024]
- padding > Expand all integer [0, 80]
- border_radius > Expand all integer [0, 60]
- overlay_text > Expand all (string | null)
- svg_path* string
- png_path > Expand all (string | null)
- created_at string date-time
- updated_at string date-time

QRPreviewResponse ^ Collapse all object

- svg_data* string
- png_data* string
- *Example* > Expand all object

Token ^ Collapse all object

- access_token* string
- token_type > Expand all string
- *Example* > Expand all object

UserCreate ^ Collapse all object

- email* string email
- full_name > Expand all (string | null)
- password* string
- *Example* > Expand all object

UserLogin ^ Collapse all object

- email* string email
- password* string
- *Example* > Expand all object

UserRead ^ Collapse all object

- email* string email

full_name > Expand all (string | null)

id* integer

created_at* string date-time

updated_at* string date-time

Example > Expand all object

UserUpdate ^ Collapse all object

full_name > Expand all (string | null)

password > Expand all (string | null)

ValidationError ^ Collapse all object

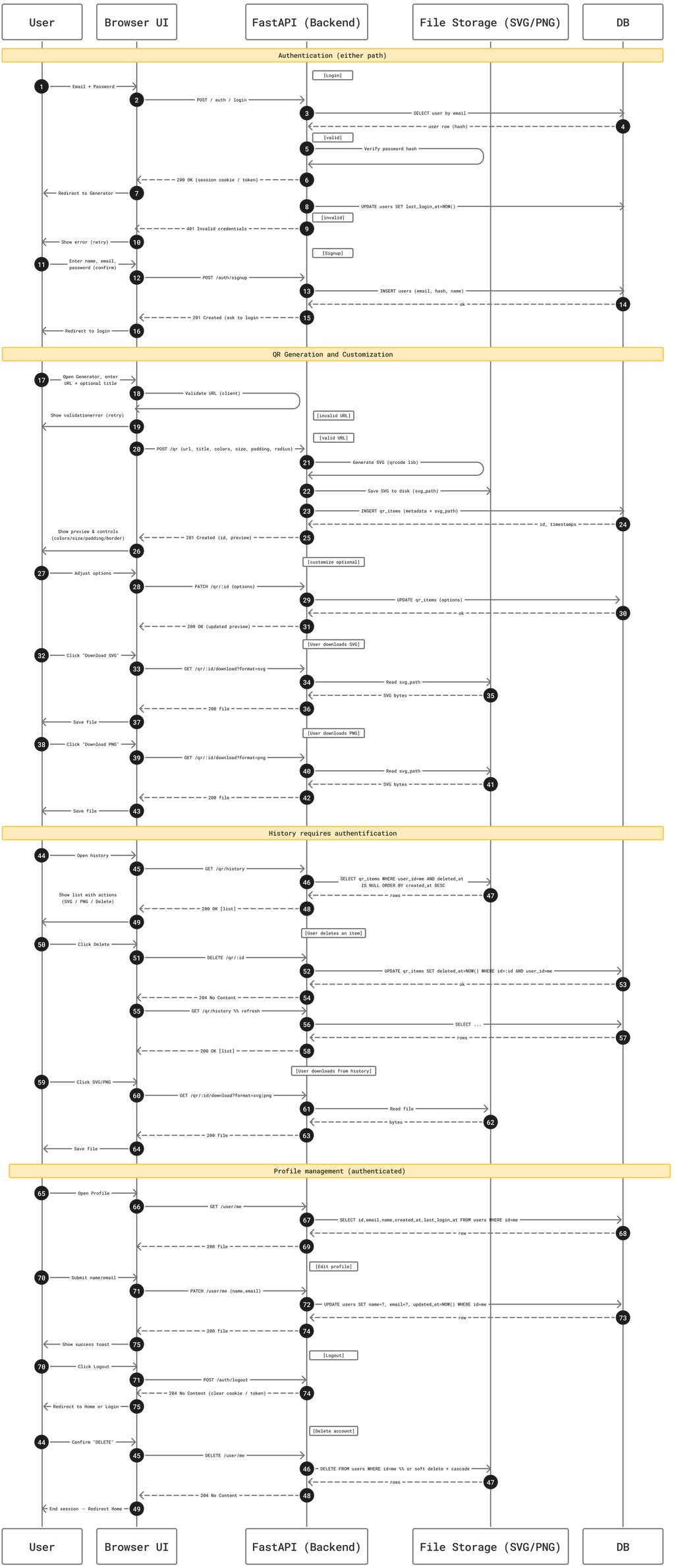
loc* > Expand all array<(string | integer)>

msg* string

type* string

Architecture Sequence Diagram

Architecture Sequence Diagram



Application Data Model

Application Data Model (ERD)

Software Development and Devops - 2025

Data Model

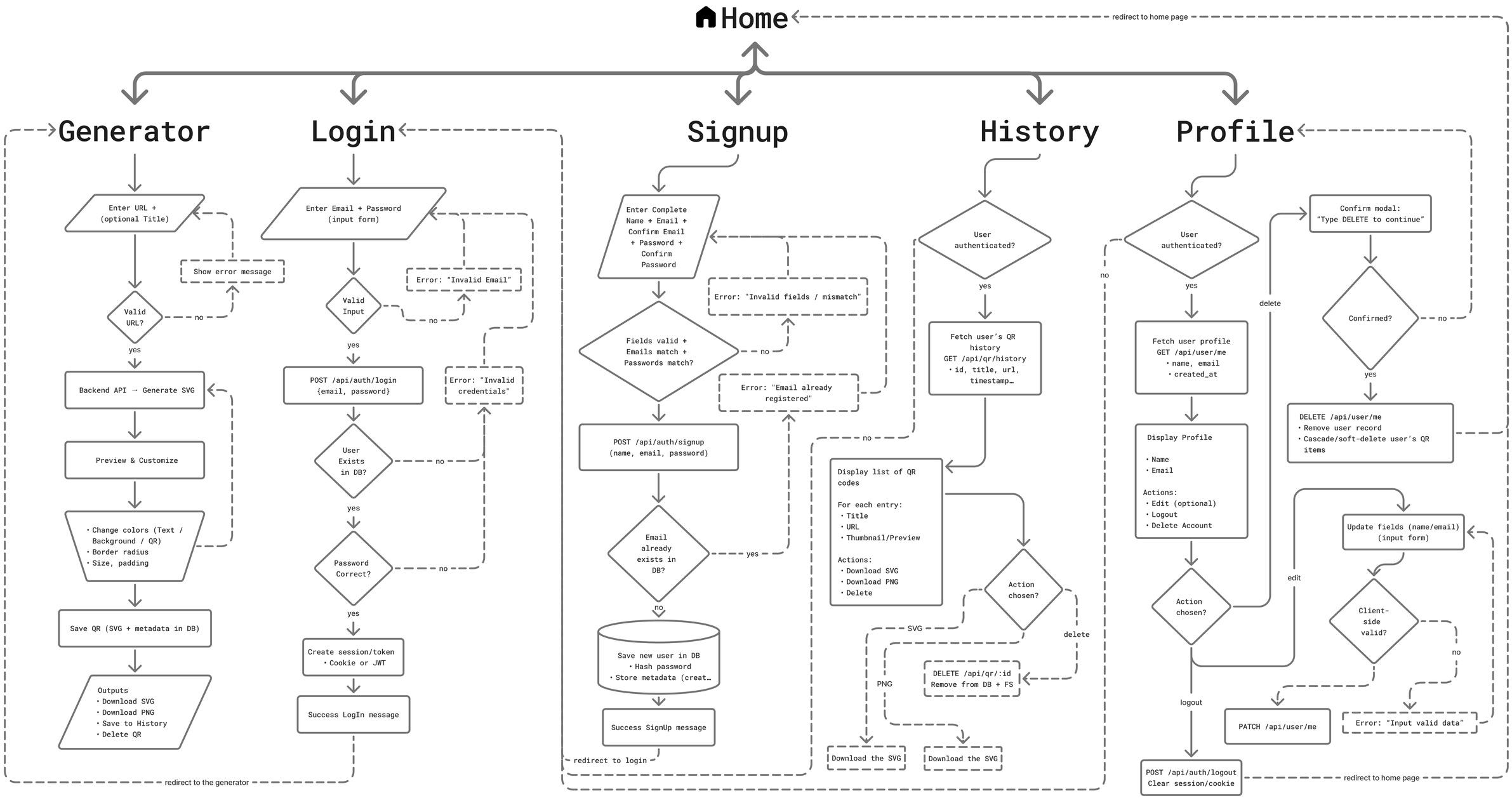
Juan Sebastian Peña Donneys

Assignment 1

Application Flow Diagram

Application Flow Diagram

Flow Diagram



Codebase

Project Codebase

Root

app.py

```
from pathlib import Path

from fastapi import FastAPI, Request
from fastapi.responses import HTMLResponse
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates

from db import init_db
from routers import auth, export, qr, user

BASE_DIR = Path(__file__).parent

TAGS_METADATA = [
    {
        "name": "auth",
        "description": "Authentication endpoints for signing up, logging in, and
logging out.",
    },
    {
        "name": "users",
        "description": "Profile management endpoints for viewing, updating, or
deleting the current user.",
    },
    {
        "name": "qr",
        "description": "QR generation, preview, download, and history endpoints
scoped to the authenticated user.",
    },
    {
        "name": "export",
        "description": "CSV export of the authenticated user's QR history.",
    },
]

app = FastAPI(
    title="QR Forge",
    description="Generate, preview, customise, and manage QR codes locally with
FastAPI.",
    version="1.0.0",
    contact={
        "name": "QR Forge",
        "url": "https://github.com/",
        "email": "support@example.com",
    },
```

```
    license_info={"name": "MIT", "url": "https://opensource.org/licenses/MIT"},
    openapi_tags=TAGS_METADATA,
    docs_url="/docs",
    redoc_url=None,
)

init_db()
app.include_router(auth.router)
app.include_router(user.router)
app.include_router(qr.router)
app.include_router(export.router)

app.mount("/assets", StaticFiles(directory="assets"), name="assets")
app.mount("/static", StaticFiles(directory="static"), name="static")


__all__ = ("app",)

templates = Jinja2Templates(directory=str(BASE_DIR / "templates"))


@app.get("/", response_class=HTMLResponse, name="home", summary="Serve the landing
page")
def home(request: Request) -> HTMLResponse:
    return templates.TemplateResponse(
        "home.html",
        {"request": request, "active_page": "home"},
    )


@app.get("/generator", response_class=HTMLResponse, name="generator_page",
summary="Serve the QR generator UI")
def generator_page(request: Request) -> HTMLResponse:
    return templates.TemplateResponse(
        "index.html",
        {"request": request, "active_page": "generator"},
    )


@app.get("/history", response_class=HTMLResponse, name="history_page",
summary="Serve the saved history UI")
def history_page(request: Request) -> HTMLResponse:
    return templates.TemplateResponse(
        "history.html",
        {"request": request, "active_page": "history"},
    )


@app.get("/profile", response_class=HTMLResponse, name="profile_page",
summary="Serve the profile management UI")
def profile_page(request: Request) -> HTMLResponse:
    return templates.TemplateResponse(
        "profile.html",
        {"request": request, "active_page": "profile"},
```

```
    )


@app.get("/login", response_class=HTMLResponse, name="login_page", summary="Serve
the login UI")
def login_page(request: Request) -> HTMLResponse:
    return templates.TemplateResponse(
        "login.html",
        {"request": request, "active_page": "login"},
    )


@app.get("/signup", response_class=HTMLResponse, name="signup_page",
summary="Serve the signup UI")
def signup_page(request: Request) -> HTMLResponse:
    return templates.TemplateResponse(
        "signup.html",
        {"request": request, "active_page": "signup"},
    )


@app.get("/health", summary="Simple health check")
def health() -> dict:
    return {"status": "ok"}
```

config.py

```
from dataclasses import dataclass
import os


@dataclass
class Settings:
    secret_key: str = os.getenv("QR_FORGE_SECRET_KEY", "change-me-in-env")
    access_token_expire_minutes: int =
int(os.getenv("QR_FORGE_TOKEN_EXPIRE_MINUTES", "720"))
    algorithm: str = os.getenv("QR_FORGE_TOKEN_ALG", "HS256")


settings = Settings()
```

core

init.py

security.py

```
from datetime import datetime, timedelta, timezone
from functools import lru_cache
from typing import Optional

import bcrypt
from fastapi import Depends, HTTPException, status
from fastapi.security import HTTPAuthorizationCredentials, HTTPBearer
from jose import JWTError, jwt
from sqlmodel import Session

from config import settings
from db import get_session
from models import User

http_bearer = HTTPBearer(auto_error=False)


class _AuthError(HTTPException):
    """Customised HTTP exception for authentication failures."""

    def __init__(self, detail: str) -> None:
        super().__init__(status_code=status.HTTP_401_UNAUTHORIZED, detail=detail)


def verify_password(plain_password: str, hashed_password: str) -> bool:
    """Return True when the provided password matches the stored bcrypt hash."""

    return bcrypt.checkpw(plain_password.encode("utf-8"),
hashed_password.encode("utf-8"))


def get_password_hash(password: str) -> str:
    """Hash the provided password using bcrypt with a randomly generated salt."""

    return bcrypt.hashpw(password.encode("utf-8"), bcrypt.gensalt()).decode("utf-
8")


def create_access_token(*, subject: int, expires_delta: Optional[timedelta] =
None) -> str:
    """Create a signed JWT using the configured algorithm and expiry."""

    expire_delta = expires_delta or
timedelta(minutes=settings.access_token_expire_minutes)
    expire = datetime.now(timezone.utc) + expire_delta
    payload = {"sub": str(subject), "exp": expire}
    return jwt.encode(payload, settings.secret_key, algorithm=settings.algorithm)
```

```
@lru_cache(maxsize=128)
def _decode_access_token(token: str) -> dict:
    """Decode and cache JWT payloads to avoid repeated signature checks in a
request."""

    return jwt.decode(token, settings.secret_key, algorithms=[settings.algorithm])


async def get_current_user(
    credentials: Optional[HTTPAuthorizationCredentials] = Depends(http_bearer),
    session: Session = Depends(get_session),
) -> User:
    """Retrieve the authenticated user based on the Authorization header."""

    if not credentials:
        raise _AuthError("Not authenticated")

    token = credentials.credentials
    try:
        payload = _decode_access_token(token)
        subject = payload.get("sub")
        if subject is None:
            raise _AuthError("Invalid token payload")
        user_id = int(subject)
    except (JWTError, ValueError):
        raise _AuthError("Invalid token") from None

    user = session.get(User, user_id)
    if not user:
        raise _AuthError("User not found")

    return user
```

Root

db.py

```
from collections.abc import Generator
from typing import Any, Dict

from sqlmodel import Session, SQLModel, create_engine

DATABASE_URL = "sqlite:///qr.db"

# sqlite needs this connect arg for threaded servers
connect_args: Dict[str, Any] = {}
if DATABASE_URL.startswith("sqlite"):
    connect_args = {"check_same_thread": False}
```

```
engine = create_engine(DATABASE_URL, echo=False, connect_args=connect_args)


def init_db() -> None:
    SQLModel.metadata.create_all(engine)


def get_session() -> Generator[Session, None, None]:
    with Session(engine) as session:
        yield session
```

exportingpdf.py

```
"""Utility script to snapshot the project code into Markdown for documentation."""
import os
from pathlib import Path
from typing import Iterable

PROJECT_ROOT = Path(__file__).parent
OUTPUT_FILE = PROJECT_ROOT / "codebase.md"
INCLUDE_EXTS = {".py", ".html", ".css", ".js"}
SKIP_FOLDERS = {".git", "__pycache__", ".venv", "node_modules"}


def should_skip(path: Path) -> bool:
    return any(part in SKIP_FOLDERS for part in path.parts)


def iter_source_files(root: Path) -> Iterable[Path]:
    for file_path in sorted(root.rglob("*")):
        if should_skip(file_path):
            continue
        if file_path.is_file() and file_path.suffix in INCLUDE_EXTS:
            yield file_path


def main() -> None:
    with OUTPUT_FILE.open("w", encoding="utf-8") as out:
        out.write("# Project Codebase\n")
        current_dir: Path | None = None
        for file_path in iter_source_files(PROJECT_ROOT):
            folder = file_path.parent.relative_to(PROJECT_ROOT)
            if folder != current_dir:
                out.write(f"\n\n## {folder if folder != Path('.') else 'Root'}\n")
                current_dir = folder
            out.write(f"\n### {file_path.name}\n")
            language = file_path.suffix.lstrip('.') or 'text'
            out.write(f"```{language}\n")
            try:
                out.write(file_path.read_text(encoding="utf-8"))
```

```
            except OSError as exc:
                out.write(f"Could not read file: {exc}")
            out.write("\n```\n")

    print(f"Codebase exported to {OUTPUT_FILE}. Convert to PDF with:\n  pandoc
{OUTPUT_FILE.name} -o codebase.pdf")


if __name__ == "__main__":
    main()
```

models.py

```
from __future__ import annotations

from datetime import datetime, timezone
from typing import Optional

from pydantic import EmailStr
from sqlmodel import Field, SQLModel


def utcnow() -> datetime:
    return datetime.now(timezone.utc)


class User(SQLModel, table=True):
    __tablename__ = "users"

    id: Optional[int] = Field(default=None, primary_key=True)
    email: EmailStr = Field(index=True, sa_column_kwargs={"unique": True})
    full_name: str = Field(default="")
    hashed_password: str
    created_at: datetime = Field(default_factory=utcnow)
    updated_at: datetime = Field(default_factory=utcnow)


class QRItem(SQLModel, table=True):
    __tablename__ = "qr_items"

    id: Optional[int] = Field(default=None, primary_key=True)
    user_id: int = Field(foreign_key="users.id", index=True)
    title: Optional[str] = Field(default=None, max_length=200)
    url: str
    foreground_color: str = Field(default="#000000", max_length=7)
    background_color: str = Field(default="#FFFFFF", max_length=7)
    size: int = Field(default=256, ge=64, le=1024)
    padding: int = Field(default=10, ge=0, le=80)
    border_radius: int = Field(default=0, ge=0, le=60)
    overlay_text: Optional[str] = Field(default=None, max_length=4)
```

```
    svg_path: str
    png_path: Optional[str] = Field(default=None)
    created_at: datetime = Field(default_factory=utcnow, index=True)
    updated_at: datetime = Field(default_factory=utcnow)
```

routers

init.py

```
from . import auth, export, qr, user

__all__ = ["auth", "export", "qr", "user"]
```

auth.py

```
from datetime import datetime, timezone

from fastapi import APIRouter, Depends, HTTPException, status
from sqlmodel import Session, select

from core.security import create_access_token, get_password_hash, verify_password
from db import get_session
from models import User
from schemas import Token, UserCreate, UserLogin, UserRead

router = APIRouter(prefix="/api/auth", tags=["auth"])


@router.post(
    "/signup",
    response_model=UserRead,
    status_code=status.HTTP_201_CREATED,
    summary="Create a new user account",
    response_description="Newly created user profile",
)
def signup(payload: UserCreate, session: Session = Depends(get_session)) -> User:
    if len(payload.password) < 8:
        raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST,
detail="Password must be at least 8 characters long")
    normalized_email = payload.email.lower()
    existing = session.exec(select(User).where(User.email ==
normalized_email)).first()
    if existing:
        raise HTTPException(status_code=status.HTTP_409_CONFLICT, detail="Email
already registered")

    now = datetime.now(timezone.utc)
```

```
    user = User(
        email=normalized_email,
        full_name=payload.full_name or "",
        hashed_password=get_password_hash(payload.password),
        created_at=now,
        updated_at=now,
    )
    session.add(user)
    session.commit()
    session.refresh(user)
    return user


@router.post(
    "/login",
    response_model=Token,
    summary="Authenticate and receive an access token",
    response_description="Bearer token for subsequent requests",
)
def login(payload: UserLogin, session: Session = Depends(get_session)) -> Token:
    normalized_email = payload.email.lower()
    user = session.exec(select(User).where(User.email ==
normalized_email)).first()
    if not user or not verify_password(payload.password, user.hashed_password):
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
detail="Invalid credentials")

    token = create_access_token(subject=user.id)
    return Token(access_token=token)


@router.post("/logout", summary="Client-side logout acknowledgement")
def logout() -> dict:
    return {"ok": True}
```

export.py

```
import csv
import io
from pathlib import Path
from typing import Iterable

from fastapi import APIRouter, Depends
from fastapi.responses import StreamingResponse
from sqlmodel import Session, select

from core.security import get_current_user
from db import get_session
from models import QRItem, User
```

```python
router = APIRouter(prefix="/api/export", tags=["export"])


@router.get(
    "/csv",
    summary="Export the authenticated user's QR history as CSV",
    response_description="CSV stream containing saved QR metadata",
)
def export_csv(
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_user),
) -> StreamingResponse:
    rows = session.exec(
        select(QRItem).where(QRItem.user_id ==
current_user.id).order_by(QRItem.created_at.desc())
    ).all()

    buf = io.StringIO()
    writer = csv.writer(buf)
    writer.writerow(
        [
            "title",
            "url",
            "created_at",
            "foreground_color",
            "background_color",
            "size",
            "padding",
            "border_radius",
            "svg_file",
            "png_file",
        ]
    )
    for r in rows:
        writer.writerow(
            [
                r.title,
                r.url,
                r.created_at.isoformat(),
                r.foreground_color,
                r.background_color,
                r.size,
                r.padding,
                r.border_radius,
                Path(r.svg_path).name if r.svg_path else "",
                Path(r.png_path).name if r.png_path else "",
            ]
        )
    buf.seek(0)

    def iter_buf() -> Iterable[str]:
        yield buf.read()

    return StreamingResponse(
```

```
        iter_buf(),
        media_type="text/csv",
        headers={"Content-Disposition": "attachment; filename=qr_history.csv"},
    )
```

qr.py

```
from datetime import datetime, timezone
from pathlib import Path
from typing import List

from fastapi import APIRouter, Depends, HTTPException, Query, status
from fastapi.responses import FileResponse
from sqlmodel import Session, select

from core.security import get_current_user
from db import get_session
from models import QRItem, User
from schemas import QRCreate, QRPreviewResponse
from services.qr import QRConfig, encode_render, generate_qr_assets, render_qr

router = APIRouter(prefix="/api/qr", tags=["qr"])
SVG_DIR = Path("generated_svgs")
PNG_DIR = Path("generated_pngs")
SVG_DIR.mkdir(parents=True, exist_ok=True)
PNG_DIR.mkdir(parents=True, exist_ok=True)


def _ensure_owner(session: Session, user: User, item_id: int) -> QRItem:
    item = session.exec(
        select(QRItem).where(QRItem.id == item_id, QRItem.user_id == user.id)
    ).first()
    if not item:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="QR item
not found")
    return item


def _to_config(payload: QRCreate) -> QRConfig:
    return QRConfig(
        url=str(payload.url),
        foreground_color=payload.foreground_color,
        background_color=payload.background_color,
        size=payload.size,
        padding=payload.padding,
        border_radius=payload.border_radius,
    )


@router.post(
```

```
    "/preview",
    response_model=QRPreviewResponse,
    summary="Render a customised QR preview without saving",
    response_description="Inline base64 PNG and SVG markup",
)
def preview_qr(
    payload: QRCreate,
    current_user: User = Depends(get_current_user),
) -> QRPreviewResponse:
    _ = current_user
    render = render_qr(_to_config(payload))
    preview = encode_render(render)
    return QRPreviewResponse(svg_data=preview.svg_data, png_data=preview.png_data)


@router.post(
    "",
    response_model=QRItem,
    status_code=status.HTTP_201_CREATED,
    summary="Persist a customised QR code",
    response_description="Saved QR item with asset paths",
)
def create_qr(
    payload: QRCreate,
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_user),
) -> QRItem:
    now = datetime.now(timezone.utc)
    assets = generate_qr_assets(_to_config(payload), svg_dir=SVG_DIR,
png_dir=PNG_DIR)

    item = QRItem(
        user_id=current_user.id,
        title=payload.title.strip() or "Untitled QR",
        url=str(payload.url),
        foreground_color=payload.foreground_color,
        background_color=payload.background_color,
        size=payload.size,
        padding=payload.padding,
        border_radius=payload.border_radius,
        overlay_text=None,
        svg_path=str(assets.svg_path),
        png_path=str(assets.png_path),
        created_at=now,
        updated_at=now,
    )
    session.add(item)
    session.commit()
    session.refresh(item)
    return item


@router.get(
    "",
```

```
    response_model=List[QRItem],
    summary="List QR codes owned by the authenticated user",
)
def list_qr(
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_user),
) -> List[QRItem]:
    return session.exec(
        select(QRItem)
        .where(QRItem.user_id == current_user.id)
        .order_by(QRItem.created_at.desc())
    ).all()


@router.get(
    "/history",
    response_model=List[QRItem],
    summary="Alias for listing QR history",
)
def history(
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_user),
) -> List[QRItem]:
    return list_qr(session=session, current_user=current_user)


@router.delete(
    "/{item_id}",
    summary="Delete a saved QR code",
    response_description="Confirmation payload",
)
def delete_qr(
    item_id: int,
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_user),
) -> dict:
    item = _ensure_owner(session, current_user, item_id)
    try:
        if item.svg_path and Path(item.svg_path).exists():
            Path(item.svg_path).unlink()
        if item.png_path and Path(item.png_path).exists():
            Path(item.png_path).unlink()
    finally:
        session.delete(item)
        session.commit()
    return {"ok": True}


@router.get(
    "/{item_id}/download",
    summary="Download a saved QR code",
    response_description="Binary SVG or PNG stream",
)
def download_qr(
```

```
    item_id: int,
    format: str = Query(default="svg", pattern="^(svg|png)$"),
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_user),
):
    item = _ensure_owner(session, current_user, item_id)
    if format == "svg":
        if not item.svg_path or not Path(item.svg_path).exists():
            raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="SVG
not available")
        return FileResponse(item.svg_path, media_type="image/svg+xml",
filename=f"qr-{item.id}.svg")

    if not item.png_path:
        raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail="PNG
export not available yet")
    if not Path(item.png_path).exists():
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="PNG not
available")
    return FileResponse(item.png_path, media_type="image/png", filename=f"qr-
{item.id}.png")
```

user.py

```
from datetime import datetime, timezone
from pathlib import Path

from fastapi import APIRouter, Depends, HTTPException, status
from sqlmodel import Session, select

from core.security import get_current_user, get_password_hash
from db import get_session
from models import QRItem, User
from schemas import UserRead, UserUpdate

router = APIRouter(prefix="/api/user", tags=["users"])


@router.get(
    "/me",
    response_model=UserRead,
    summary="Return the authenticated user's profile",
    response_description="Current user record",
)
def read_current_user(current_user: User = Depends(get_current_user)) -> User:
    return current_user


@router.patch(
    "/me",
```

```
    response_model=UserRead,
    summary="Update profile details",
    response_description="Updated user record",
)
def update_current_user(
    payload: UserUpdate,
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_user),
) -> User:
    updated = False
    if payload.full_name is not None:
        current_user.full_name = payload.full_name.strip()
        updated = True
    if payload.password:
        if len(payload.password) < 8:
            raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST,
detail="Password must be at least 8 characters long")
        current_user.hashed_password = get_password_hash(payload.password)
        updated = True
    if not updated:
        return current_user
    current_user.updated_at = datetime.now(timezone.utc)
    session.add(current_user)
    session.commit()
    session.refresh(current_user)
    return current_user


@router.delete(
    "/me",
    summary="Delete the authenticated user and all owned QR codes",
    response_description="Confirmation payload",
)
def delete_current_user(
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_user),
) -> dict:
    qrs = session.exec(select(QRItem).where(QRItem.user_id ==
current_user.id)).all()
    for item in qrs:
        if item.svg_path and Path(item.svg_path).exists():
            Path(item.svg_path).unlink()
        if item.png_path and Path(item.png_path).exists():
            Path(item.png_path).unlink()
        session.delete(item)
    session.delete(current_user)
    session.commit()
    return {"ok": True}
```

Root

schemas.py

```
from datetime import datetime
from typing import Annotated, Optional

from pydantic import BaseModel, ConfigDict, EmailStr, Field, HttpUrl


HexColor = Annotated[str, Field(pattern=r"^#[0-9a-fA-F]{6}$")]
HexOrTransparent = Annotated[str, Field(pattern=r"^(#[0-9a-fA-F]
{6}|transparent)$")]


class UserBase(BaseModel):
    email: EmailStr
    full_name: Optional[str] = None


class UserCreate(UserBase):
    password: str

    model_config = ConfigDict(json_schema_extra={
        "example": {
            "email": "user@example.com",
            "full_name": "Sample User",
            "password": "strongpass123",
        }
    })


class UserRead(UserBase):
    id: int
    created_at: datetime
    updated_at: datetime

    model_config = ConfigDict(json_schema_extra={
        "example": {
            "id": 1,
            "email": "user@example.com",
            "full_name": "Sample User",
            "created_at": "2024-01-01T12:00:00Z",
            "updated_at": "2024-01-02T12:00:00Z",
        }
    })


class UserLogin(BaseModel):
    email: EmailStr
    password: str

    model_config = ConfigDict(json_schema_extra={
        "example": {
```

```
            "email": "user@example.com",
            "password": "strongpass123",
        }
    })


class UserUpdate(BaseModel):
    full_name: Optional[str] = None
    password: Optional[str] = None


class Token(BaseModel):
    access_token: str
    token_type: str = "bearer"

    model_config = ConfigDict(json_schema_extra={
        "example": {
            "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
            "token_type": "bearer",
        }
    })


class TokenPayload(BaseModel):
    sub: int
    exp: datetime


class QRBase(BaseModel):
    title: str
    url: HttpUrl
    foreground_color: HexColor = "#000000"
    background_color: HexOrTransparent = "#FFFFFF"
    size: int = Field(default=512, ge=128, le=1024)
    padding: int = Field(default=16, ge=0, le=128)
    border_radius: int = Field(default=0, ge=0, le=120)

    model_config = ConfigDict(json_schema_extra={
        "example": {
            "title": "Campaign QR",
            "url": "https://example.com",
            "foreground_color": "#1f3a93",
            "background_color": "#ffffff",
            "size": 320,
            "padding": 12,
            "border_radius": 20,
        }
    })


class QRCreate(QRBase):
    pass
```

```
class QRPreviewResponse(BaseModel):
    svg_data: str
    png_data: str

    model_config = ConfigDict(json_schema_extra={
        "example": {
            "svg_data": "<svg xmlns=...>",
            "png_data": "iVBORw0KGgoAAAANSUhEUg...",
        }
    })
```

services

init.py

```

```

qr.py

```
from __future__ import annotations

import base64
import io
import uuid
from dataclasses import dataclass
from pathlib import Path
from typing import List, Tuple

import qrcode
from PIL import Image, ImageDraw


@dataclass
class QRConfig:
    url: str
    foreground_color: str
    background_color: str
    size: int
    padding: int
    border_radius: int


@dataclass
class QRPreview:
    svg_data: str
    png_data: str
```

```python
@dataclass
class QRRender:
    svg_text: str
    png_bytes: bytes


@dataclass
class QRAssets:
    svg_path: Path
    png_path: Path


HEX_ALPHA = 255
TRANSPARENT = (0, 0, 0, 0)


def _ensure_dir(path: Path) -> Path:
    path.mkdir(parents=True, exist_ok=True)
    return path


def _hex_to_rgba(color: str) -> Tuple[int, int, int, int]:
    if color.lower() == 'transparent':
        return TRANSPARENT
    color = color.lstrip('#')
    if len(color) != 6:
        raise ValueError('Expected 6 character hex color or "transparent"')
    r = int(color[0:2], 16)
    g = int(color[2:4], 16)
    b = int(color[4:6], 16)
    return r, g, b, HEX_ALPHA


def _create_matrix(config: QRConfig) -> List[List[bool]]:
    qr = qrcode.QRCode(
        version=None,
        error_correction=qrcode.constants.ERROR_CORRECT_M,
        border=0,
    )
    qr.add_data(config.url)
    qr.make(fit=True)
    return qr.get_matrix()


def _render_svg(config: QRConfig, matrix: List[List[bool]]) -> str:
    modules = len(matrix)
    module_size = config.size / modules
    total_size = config.size + config.padding * 2
    bg = config.background_color
    svg_parts = [
        f'<svg xmlns="http://www.w3.org/2000/svg" width="{total_size}" height="
{total_size}" viewBox="0 0 {total_size} {total_size}">'
    ]
    if bg.lower() != 'transparent':
```

```
        svg_parts.append(
            f'<rect width="{total_size}" height="{total_size}" fill="{bg}" rx="
{config.border_radius}" ry="{config.border_radius}" />'
        )
    pad = config.padding
    fg = config.foreground_color
    for y, row in enumerate(matrix):
        for x, cell in enumerate(row):
            if not cell:
                continue
            x0 = pad + x * module_size
            y0 = pad + y * module_size
            svg_parts.append(
                f'<rect x="{x0:.3f}" y="{y0:.3f}" width="{module_size:.3f}"
height="{module_size:.3f}" fill="{fg}" />'
            )
    svg_parts.append('</svg>')
    return ''.join(svg_parts)


def _render_png(config: QRConfig, matrix: List[List[bool]]) -> bytes:
    modules = len(matrix)
    module_size = config.size / modules
    total_size = config.size + config.padding * 2

    background = Image.new('RGBA', (total_size, total_size),
_hex_to_rgba(config.background_color))
    draw = ImageDraw.Draw(background)
    fg_rgba = _hex_to_rgba(config.foreground_color)

    for y, row in enumerate(matrix):
        for x, cell in enumerate(row):
            if not cell:
                continue
            x0 = config.padding + x * module_size
            y0 = config.padding + y * module_size
            x1 = x0 + module_size
            y1 = y0 + module_size
            draw.rectangle([x0, y0, x1, y1], fill=fg_rgba)

    if config.border_radius > 0:
        radius = min(config.border_radius, total_size // 2)
        mask = Image.new('L', (total_size, total_size), 0)
        mask_draw = ImageDraw.Draw(mask)
        mask_draw.rounded_rectangle((0, 0, total_size, total_size), radius=radius,
fill=255)
        rounded = Image.new('RGBA', (total_size, total_size))
        rounded.paste(background, (0, 0), mask=mask)
        background = rounded

    with io.BytesIO() as buf:
        background.save(buf, format='PNG')
        return buf.getvalue()
```

```
def render_qr(config: QRConfig) -> QRRender:
    matrix = _create_matrix(config)
    svg_text = _render_svg(config, matrix)
    png_bytes = _render_png(config, matrix)
    return QRRender(svg_text=svg_text, png_bytes=png_bytes)


def generate_qr_assets(
    config: QRConfig,
    *,
    svg_dir: Path,
    png_dir: Path,
) -> QRAssets:
    svg_dir = _ensure_dir(svg_dir)
    png_dir = _ensure_dir(png_dir)
    render = render_qr(config)

    svg_filename = f"{uuid.uuid4()}.svg"
    svg_path = svg_dir / svg_filename
    svg_path.write_text(render.svg_text, encoding='utf-8')

    png_filename = f"{svg_filename[:-4]}.png"
    png_path = png_dir / png_filename
    png_path.write_bytes(render.png_bytes)

    return QRAssets(svg_path=svg_path, png_path=png_path)


def encode_render(render: QRRender) -> QRPreview:
    return QRPreview(
        svg_data=render.svg_text,
        png_data=base64.b64encode(render.png_bytes).decode('ascii'),
    )
```

static

app.js

```
const storageKey = 'qrforge_token';
const authState = {
  token: localStorage.getItem(storageKey) || null,
};

const body = document.body;
const toastEl = document.getElementById('toast');

function toast(message, duration = 2200) {
  if (!toastEl) return;
  toastEl.textContent = message;
```

```
  toastEl.classList.add('show');
  setTimeout(() => toastEl.classList.remove('show'), duration);
}

function isAuthed() {
  return Boolean(authState.token);
}

function updateAuthStateUI() {
  const authed = isAuthed();
  body?.classList.toggle('authed', authed);
  document.querySelectorAll('[data-requires-auth]').forEach((el) => {
    el.classList.toggle('hidden', !authed);
  });
  document.querySelectorAll('[data-hide-when-authed]').forEach((el) => {
    el.classList.toggle('hidden', authed);
  });
}

function broadcastAuthState() {
  document.dispatchEvent(new CustomEvent('auth-change', { detail: { authed:
isAuthed() } }));
}

function setToken(token) {
  if (token) {
    localStorage.setItem(storageKey, token);
    authState.token = token;
  } else {
    localStorage.removeItem(storageKey);
    authState.token = null;
  }
  updateAuthStateUI();
  broadcastAuthState();
}

updateAuthStateUI();
broadcastAuthState();

let unauthorizedNoticeShown = false;
function handleUnauthorized() {
  if (authState.token) setToken(null);
  if (!unauthorizedNoticeShown) {
    unauthorizedNoticeShown = true;
    toast('Please log in to continue');
    setTimeout(() => {
      unauthorizedNoticeShown = false;
      window.location.href = '/login';
    }, 400);
  }
  throw new Error('Unauthorized');
}

function authorizedFetch(url, options = {}) {
```

```
  if (!authState.token) throw new Error('Unauthorized');
  const opts = { ...options };
  const headers = new Headers(options.headers || {});
  headers.set('Authorization', `Bearer ${authState.token}`);
  opts.headers = headers;
  return fetch(url, opts).then((res) => {
    if (res.status === 401) handleUnauthorized();
    return res;
  });
}

function requireAuth() {
  if (isAuthed()) return true;
  toast('Please log in to continue');
  setTimeout(() => {
    window.location.href = '/login';
  }, 400);
  return false;
}

function escapeHtml(value) {
  return String(value ?? '').replace(/[&<>"']/g, (match) => ({
    '&': '&amp;',
    '<': '&lt;',
    '>': '&gt;',
    '"': '&quot;',
    "'": '&#39;',
  })[match]);
}

function formatDate(value) {
  if (!value) return '—';
  try {
    return new Date(value).toLocaleString();
  } catch (err) {
    return value;
  }
}

function sanitizeFilename(title, ext) {
  const safe = String(title || 'qr-code')
    .toLowerCase()
    .replace(/[^a-z0-9]+/g, '-')
    .replace(/^-+|-+$/g, '') || 'qr-code';
  return `${safe}.${ext}`;
}

function triggerDownload(blob, filename) {
  const url = URL.createObjectURL(blob);
  const link = document.createElement('a');
  link.href = url;
  link.download = filename;
  document.body.appendChild(link);
  link.click();
```

```
  link.remove();
  URL.revokeObjectURL(url);
}

function base64ToBlob(base64, type) {
  const binary = atob(base64);
  const bytes = new Uint8Array(binary.length);
  for (let i = 0; i < binary.length; i += 1) bytes[i] = binary.charCodeAt(i);
  return new Blob([bytes], { type });
}

const historyTargets = {
  drawer: document.getElementById('historyList'),
  page: document.getElementById('historyGrid'),
  empty: document.getElementById('historyEmpty'),
  guard: document.getElementById('historyGuard'),
  content: document.getElementById('historyContent'),
};

let historyCache = new Map();

function historyCardTemplate(item) {
  const created = formatDate(item.created_at);
  const title = escapeHtml(item.title || 'Untitled QR');
  // Placeholder src, will be replaced after fetch
  return `
    <article class="history-card" data-id="${item.id}" data-title="${title}">
      <img data-thumb-id="${item.id}" alt="Preview for ${title}" loading="lazy" />
      <div class="history-card-body">
        <div class="history-card-title">${title}</div>
        <div class="history-card-meta">Created ${created}</div>
      </div>
      <div class="history-card-actions">
        <button class="btn ghost small" type="button" data-action="svg" data-
id="${item.id}">SVG</button>
        <button class="btn small" type="button" data-action="png" data-
id="${item.id}">PNG</button>
        <button class="btn danger small" type="button" data-action="delete" data-
id="${item.id}">Delete</button>
      </div>
    </article>
  `;
}

function bindHistoryActions(container) {
  if (!container) return;
  container.querySelectorAll('[data-action]').forEach((btn) => {
    btn.addEventListener('click', async () => {
      const action = btn.dataset.action;
      const id = btn.dataset.id;
      if (!id) return;
      const item = historyCache.get(id);
      if (!item) return;
      if (action === 'delete') {
```

```
        if (!requireAuth()) return;
        const confirmed = confirm('Delete this QR?');
        if (!confirmed) return;
        try {
          await authorizedFetch(`/api/qr/${id}`, { method: 'DELETE' });
          toast('QR deleted');
          await loadHistory();
        } catch (err) {
          if (err.message !== 'Unauthorized') {
            console.error(err);
            toast('Unable to delete QR');
          }
        }
      } else if (action === 'png' || action === 'svg') {
        if (!requireAuth()) return;
        downloadAsset(item, action);
      }
    });
  });
}

async function updateHistoryUI(items) {
  historyCache = new Map(items.map((entry) => [String(entry.id), entry]));
  const hasItems = items.length > 0;
  historyTargets.empty?.classList.toggle('hidden', hasItems);
  if (historyTargets.page) {
    historyTargets.page.classList.toggle('empty', !hasItems);
    historyTargets.page.innerHTML = hasItems ?
items.map(historyCardTemplate).join('') : '';
    bindHistoryActions(historyTargets.page);
    // Fetch and set thumbnails
    if (hasItems) {
      for (const item of items) {
        try {
          const res = await authorizedFetch(`/api/qr/${item.id}/download?
format=png&v=${encodeURIComponent(item.updated_at || '')}`);
          if (res.ok) {
            const blob = await res.blob();
            const url = URL.createObjectURL(blob);
            const img = historyTargets.page.querySelector(`img[data-thumb-
id="${item.id}"]`);
            if (img) img.src = url;
          }
        } catch (err) { /* ignore */ }
      }
    }
  }
  if (historyTargets.drawer) {
    historyTargets.drawer.innerHTML = hasItems
      ? items.slice(0, 8).map(historyCardTemplate).join('')
      : '<div class="history-empty">No QR codes yet. Generate a new one to see it
here.</div>';
    bindHistoryActions(historyTargets.drawer);
    // Fetch and set thumbnails for drawer
```

```
    if (hasItems) {
      for (const item of items.slice(0, 8)) {
        try {
          const res = await authorizedFetch(`/api/qr/${item.id}/download?
format=png&v=${encodeURIComponent(item.updated_at || '')}`);
          if (res.ok) {
            const blob = await res.blob();
            const url = URL.createObjectURL(blob);
            const img = historyTargets.drawer.querySelector(`img[data-thumb-
id="${item.id}"]`);
            if (img) img.src = url;
          }
        } catch (err) { /* ignore */ }
      }
    }
  }
}

async function loadHistory() {
  const authed = isAuthed();
  historyTargets.guard?.classList.toggle('hidden', authed);
  historyTargets.content?.classList.toggle('hidden', !authed);
  if (!authed) {
    if (historyTargets.drawer) {
      historyTargets.drawer.innerHTML = '<div class="history-empty">Login to see
your saved QR codes.</div>';
    }
    historyCache.clear();
    return [];
  }
  try {
    const res = await authorizedFetch('/api/qr/history');
    if (!res.ok) throw new Error('Failed to load history');
    const items = await res.json();
    updateHistoryUI(items);
    return items;
  } catch (err) {
    if (err.message !== 'Unauthorized') {
      console.error(err);
      toast('Unable to load history');
    }
    return [];
  }
}

async function downloadCsv() {
  if (!requireAuth()) return;
  try {
    const res = await authorizedFetch('/api/export/csv');
    if (!res.ok) throw new Error('Failed to export history');
    const blob = await res.blob();
    triggerDownload(blob, 'qr-history.csv');
    toast('History exported');
  } catch (err) {
```

```
    if (err.message !== 'Unauthorized') {
      console.error(err);
      toast('Unable to export history');
    }
  }
}

async function downloadAsset(item, format) {
  try {
    const res = await authorizedFetch(`/api/qr/${item.id}/download?
format=${format}`);
    if (!res.ok) throw new Error('Download failed');
    const blob = await res.blob();
    triggerDownload(blob, sanitizeFilename(item.title, format));
  } catch (err) {
    if (err.message !== 'Unauthorized') {
      console.error(err);
      toast('Unable to download file');
    }
  }
}

async function fetchAssetBlob(id, format) {
  const res = await authorizedFetch(`/api/qr/${id}/download?format=${format}`);
  if (!res.ok) throw new Error('Download failed');
  return res.blob();
}

function initHistory() {
  document.getElementById('refreshHistory')?.addEventListener('click', () =>
loadHistory());
  document.getElementById('exportCsv')?.addEventListener('click', () =>
downloadCsv());
  if (historyTargets.drawer || historyTargets.page || historyTargets.guard) {
    loadHistory();
    document.addEventListener('auth-change', () => loadHistory());
  }
}

function payloadFromForm(form) {
  const transparent = form.querySelector('#bgTransparent')?.checked;
  return {
    title: (form.querySelector('#title')?.value || '').trim(),
    url: form.querySelector('#url')?.value?.trim(),
    foreground_color: form.querySelector('#fgColor')?.value || '#000000',
    background_color: transparent ? 'transparent' :
form.querySelector('#bgColor')?.value || '#ffffff',
    size: Number(form.querySelector('#sizeRange')?.value || 512),
    padding: Number(form.querySelector('#paddingRange')?.value || 16),
    border_radius: Number(form.querySelector('#radiusRange')?.value || 0),
  };
}

function payloadsMatch(a, b) {
```

```
  if (!a || !b) return false;
  return (
    a.title === b.title &&
    a.url === b.url &&
    a.foreground_color === b.foreground_color &&
    a.background_color === b.background_color &&
    a.size === b.size &&
    a.padding === b.padding &&
    a.border_radius === b.border_radius
  );
}

function initGenerator() {
  if (!body.classList.contains('page-generator')) return;

  const guard = document.getElementById('generatorGuard');
  const generatorNodes = document.querySelectorAll('[data-generator-auth]');
  const form = document.getElementById('qr-form');
  const pdf = document.getElementById('qrBox');
  const qrBox = document.getElementById('qrBox');
  const previewImg = document.getElementById('qrPreview');
  const previewEmpty = document.getElementById('qrEmpty');
  const fgColor = document.getElementById('fgColor');
  const bgColor = document.getElementById('bgColor');
  const bgTransparent = document.getElementById('bgTransparent');
  const sizeRange = document.getElementById('sizeRange');
  const paddingRange = document.getElementById('paddingRange');
  const radiusRange = document.getElementById('radiusRange');
  const urlInput = document.getElementById('url');
  const sizeValue = document.getElementById('sizeValue');
  const paddingValue = document.getElementById('paddingValue');
  const previewBtn = document.getElementById('previewBtn');
  const saveBtn = document.getElementById('saveQr');
  const dlSvg = document.getElementById('dlSvg');
  const dlPng = document.getElementById('dlPng');
  const openHistoryBtn = document.getElementById('openHistory');
  const closeHistoryBtn = document.getElementById('closeHistory');
  const historyDrawer = document.getElementById('historyDrawer');

  const formElements = form ? Array.from(form.querySelectorAll('input, button')) :
[];
  const controlInputs = [fgColor, bgColor, bgTransparent, sizeRange, paddingRange,
radiusRange, urlInput];

  let lastPreview = null;
  let lastSaved = null;
  let previewDebounce = null;

  function setGeneratorAuthState(authed) {
    guard?.classList.toggle('hidden', authed);
    generatorNodes.forEach((node) => node.classList.toggle('hidden', !authed));
    formElements.forEach((el) => (el.disabled = !authed));
    controlInputs.forEach((el) => {
      if (el) el.disabled = !authed;
```

```
    });
    saveBtn.disabled = !authed;
    if (!authed) {
      historyDrawer?.classList.remove('open');
      lastPreview = null;
      lastSaved = null;
      previewImg?.setAttribute('style', 'display:none');
      previewImg?.removeAttribute('src');
      previewEmpty?.classList.remove('hidden');
      if (qrBox) {
        qrBox.style.backgroundColor = 'transparent';
        qrBox.style.borderRadius = '20px';
      }
    }
  }

  function applyControlLabels() {
    if (sizeValue && sizeRange) sizeValue.textContent = `${sizeRange.value} px`;
    if (paddingValue && paddingRange) paddingValue.textContent =
`${paddingRange.value} px`;
  }

  function applyPreviewStyles(payload) {
    if (!qrBox || !previewImg) return;
    const bg = payload.background_color === 'transparent' ? 'transparent' :
payload.background_color;
    qrBox.style.backgroundColor = bg;
    qrBox.style.borderRadius = `${payload.border_radius}px`;
    previewImg.style.borderRadius = `${Math.max(payload.border_radius - 4, 0)}px`;
  }

  function setPreviewFromBase64(pngData, payload) {
    if (!pngData || !previewImg) return;
    const dataUrl = `${pngData}`;
    previewImg.src = dataUrl;
    previewImg.style.display = 'block';
    previewEmpty?.classList.add('hidden');
    applyPreviewStyles(payload);
  }

  async function requestPreview(payload) {
    if (!isAuthed()) return null;
    if (!payload.url) return null;
    try {
      new URL(payload.url);
    } catch (err) {
      return null;
    }
    const res = await authorizedFetch('/api/qr/preview', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(payload),
    });
    if (!res.ok) throw new Error(await res.text());
```

```
  return res.json();
}

async function handlePreview(payload) {
  try {
    const preview = await requestPreview(payload);
    if (!preview) return;
    setPreviewFromBase64(preview.png_data, payload);
    lastPreview = {
      payload,
      pngData: preview.png_data,
      svg: preview.svg_data,
    };
    if (!lastSaved || !payloadsMatch(lastSaved.payload, payload)) {
      lastSaved = null;
    }
    saveBtn.disabled = false;
  } catch (err) {
    if (err.message !== 'Unauthorized') {
      console.error(err);
      toast('Unable to preview QR');
    }
  }
}

function schedulePreview(payload) {
  if (!isAuthed()) return;
  if (previewDebounce) clearTimeout(previewDebounce);
  previewDebounce = setTimeout(() => {
    handlePreview(payload);
  }, 250);
}

form?.addEventListener('submit', (event) => {
  event.preventDefault();
  if (!requireAuth() || !form) return;
  const payload = payloadFromForm(form);
  if (!payload.url) {
    toast('Please enter a valid URL');
    return;
  }
  previewBtn.disabled = true;
  handlePreview(payload).finally(() => {
    previewBtn.disabled = false;
  });
});

saveBtn?.addEventListener('click', async () => {
  if (!lastPreview || !requireAuth()) return;
  saveBtn.disabled = true;
  try {
    const res = await authorizedFetch('/api/qr', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
```

```
        body: JSON.stringify(lastPreview.payload),
      });
      if (!res.ok) throw new Error(await res.text());
      const item = await res.json();
      lastSaved = { payload: lastPreview.payload, item };
      toast('QR saved to history');
      loadHistory();
    } catch (err) {
      if (err.message !== 'Unauthorized') {
        console.error(err);
        toast('Unable to save QR');
      }
    } finally {
      saveBtn.disabled = false;
    }
  });

  [fgColor, bgColor, bgTransparent, sizeRange, paddingRange, radiusRange,
urlInput].forEach((input) => {
    input?.addEventListener('input', () => {
      applyControlLabels();
      if (!form) return;
      if (input === bgTransparent) {
        bgColor.disabled = bgTransparent.checked;
      }
      const payload = payloadFromForm(form);
      if (!payload.url) return;
      schedulePreview(payload);
    });
  });

  if (bgTransparent?.checked) {
    bgColor.disabled = true;
  }

  dlSvg?.addEventListener('click', () => {
    if (!lastPreview) {
      toast('Preview a QR code first');
      return;
    }
    if (lastSaved && payloadsMatch(lastSaved.payload, lastPreview.payload)) {
      if (!requireAuth()) return;
      downloadAsset(lastSaved.item, 'svg');
      return;
    }
    const blob = new Blob([lastPreview.svg], { type: 'image/svg+xml' });
    triggerDownload(blob, sanitizeFilename(lastPreview.payload.title, 'svg'));
  });

  dlPng?.addEventListener('click', () => {
    if (!lastPreview) {
      toast('Preview a QR code first');
      return;
    }
```

```
    if (lastSaved && payloadsMatch(lastSaved.payload, lastPreview.payload)) {
      if (!requireAuth()) return;
      downloadAsset(lastSaved.item, 'png');
      return;
    }
    const blob = base64ToBlob(lastPreview.pngData, 'image/png');
    triggerDownload(blob, sanitizeFilename(lastPreview.payload.title, 'png'));
  });

  openHistoryBtn?.addEventListener('click', () => {
    if (!requireAuth()) return;
    historyDrawer?.classList.add('open');
    loadHistory();
  });
  closeHistoryBtn?.addEventListener('click', () =>
historyDrawer?.classList.remove('open'));
  document.addEventListener('keydown', (event) => {
    if (event.key === 'Escape') historyDrawer?.classList.remove('open');
  });

  applyControlLabels();
  setGeneratorAuthState(isAuthed());
  document.addEventListener('auth-change', (event) => {
    setGeneratorAuthState(Boolean(event.detail?.authed));
  });
}

function initAuthForms() {
  const loginForm = document.getElementById('login-form');
  if (loginForm) {
    loginForm.addEventListener('submit', async (event) => {
      event.preventDefault();
      const fd = new FormData(loginForm);
      const payload = { email: fd.get('email'), password: fd.get('password') };
      const submitBtn = loginForm.querySelector('button[type="submit"]');
      if (submitBtn) submitBtn.disabled = true;
      try {
        const res = await fetch('/api/auth/login', {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify(payload),
        });
        if (!res.ok) {
          let message = 'Unable to login';
          try {
            const error = await res.json();
            message = error.detail || message;
          } catch (err) {
            message = await res.text() || message;
          }
          toast(message);
          return;
        }
        const data = await res.json();
```

```
        setToken(data.access_token);
        toast('Welcome back!');
        window.location.href = '/generator';
      } catch (err) {
        console.error(err);
        toast('Unable to login');
      } finally {
        if (submitBtn) submitBtn.disabled = false;
      }
    });
  }

  const signupForm = document.getElementById('signup-form');
  if (signupForm) {
    signupForm.addEventListener('submit', async (event) => {
      event.preventDefault();
      const fd = new FormData(signupForm);
      const payload = {
        full_name: fd.get('full_name'),
        email: fd.get('email'),
        password: fd.get('password'),
      };
      const submitBtn = signupForm.querySelector('button[type="submit"]');
      if (submitBtn) submitBtn.disabled = true;
      try {
        const res = await fetch('/api/auth/signup', {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify(payload),
        });
        if (!res.ok) {
          let message = 'Unable to sign up';
          try {
            const error = await res.json();
            message = error.detail || message;
          } catch (err) {
            message = await res.text() || message;
          }
          toast(message);
          return;
        }
        toast('Account created!');
        const loginRes = await fetch('/api/auth/login', {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({ email: payload.email, password: payload.password
}),
        });
        if (!loginRes.ok) {
          toast('Account created. Please login.');
          window.location.href = '/login';
          return;
        }
        const data = await loginRes.json();
```

```
        setToken(data.access_token);
        toast('Welcome to QR Forge!');
        window.location.href = '/generator';
      } catch (err) {
        console.error(err);
        toast('Unable to sign up');
      } finally {
        if (submitBtn) submitBtn.disabled = false;
      }
    });
  }
}

function initProfile() {
  const fullNameInput = document.getElementById('profileFullName');
  const emailInput = document.getElementById('profileEmail');
  const sinceInput = document.getElementById('profileSince');
  const guard = document.getElementById('profileGuard');
  const content = document.getElementById('profileContent');
  const logoutBtn = document.getElementById('logoutBtn');
  const deleteAccountBtn = document.getElementById('deleteAccount');
  const profileForm = document.getElementById('profileForm');
  const passwordInput = document.getElementById('profilePassword');

  if (!profileForm) return;

  async function syncProfile() {
    if (!isAuthed()) {
      guard?.classList.remove('hidden');
      content?.classList.add('hidden');
      return;
    }
    guard?.classList.add('hidden');
    content?.classList.remove('hidden');
    try {
      const res = await authorizedFetch('/api/user/me');
      if (!res.ok) throw new Error('Failed to load profile');
      const user = await res.json();
      if (fullNameInput) fullNameInput.value = user.full_name || '';
      if (emailInput) emailInput.value = user.email || '';
      if (sinceInput) sinceInput.value = formatDate(user.created_at);
      if (passwordInput) passwordInput.value = '';
    } catch (err) {
      if (err.message !== 'Unauthorized') {
        console.error(err);
        toast('Unable to load profile');
      }
    }
  }

  profileForm.addEventListener('submit', async (event) => {
    event.preventDefault();
    if (!requireAuth()) return;
    const payload = {};
```

```
    if (fullNameInput && fullNameInput.value.trim() !== '') {
      payload.full_name = fullNameInput.value.trim();
    }
    if (passwordInput && passwordInput.value) {
      payload.password = passwordInput.value;
    }
    if (Object.keys(payload).length === 0) {
      toast('Nothing to update');
      return;
    }
    const res = await authorizedFetch('/api/user/me', {
      method: 'PATCH',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(payload),
    });
    if (!res.ok) {
      const msg = await res.text();
      toast(msg || 'Unable to update profile');
      return;
    }
    toast('Profile updated');
    await syncProfile();
  });

  logoutBtn?.addEventListener('click', async () => {
    if (!requireAuth()) return;
    try {
      await authorizedFetch('/api/auth/logout', { method: 'POST' });
    } catch (err) {
      if (err.message !== 'Unauthorized') console.warn(err);
    } finally {
      setToken(null);
      toast('Logged out');
      window.location.href = '/login';
    }
  });

  deleteAccountBtn?.addEventListener('click', async () => {
    if (!requireAuth()) return;
    const confirmed = confirm('Delete your account and all saved QR codes? This
cannot be undone.');
    if (!confirmed) return;
    try {
      const res = await authorizedFetch('/api/user/me', { method: 'DELETE' });
      if (!res.ok) throw new Error(await res.text());
      setToken(null);
      toast('Account deleted');
      window.location.href = '/signup';
    } catch (err) {
      if (err.message !== 'Unauthorized') {
        console.error(err);
        toast('Unable to delete account');
      }
    }
```

```
  });

  syncProfile();
  document.addEventListener('auth-change', () => syncProfile());
}

initAuthForms();
initProfile();
initHistory();
initGenerator();
```

styles.css

```
/* Button style for color pickers */
.color-btn {
  display: inline-block;
  background: #fff;
  color: #0d2f77;
  border-radius: 8px;
  padding: 6px 18px;
  font-weight: 600;
  cursor: pointer;
  border: 1px solid #dfe6f5;
  position: relative;
  min-width: 90px;
  text-align: center;
}
/* Style for color input to make color visible and match right-side look */
input[type="color"] {
  width: 48px;
  height: 36px;
  border: none;
  background: none;
  box-shadow: 0 1px 4px rgba(24,51,116,.10);
  border-radius: 8px;
  padding: 0;
  margin-top: 6px;
  cursor: pointer;
  display: block;
}
input[type="color"]::-webkit-color-swatch {
  border-radius: 8px;
  border: 1px solid #dfe6f5;
}
input[type="color"]::-moz-color-swatch {
  border-radius: 8px;
  border: 1px solid #dfe6f5;
}
input[type="color"]::-ms-color-swatch {
  border-radius: 8px;
```

```
  border: 1px solid #dfe6f5;
}

/* All CSS variables properly enclosed in :root */
:root {
  --primary-color: #4F8A8B;
  --secondary-color: #FBD46D;
  --accent-color: #F76B8A;
  --background-color: #F9F9F9;
  --text-color: #222831;
  --input-bg: #fff;
  --input-border: #ccc;
  --input-focus: #4F8A8B;
  --btn-bg: #4F8A8B;
  --btn-bg-hover: #357376;
  --btn-text: #fff;
  --color-picker-border: #ccc;
  --color-picker-bg: #fff;
  --color-picker-thumb: #4F8A8B;
  --color-btn-border: #ccc;
  --color-btn-bg: #fff;
  --color-btn-hover: #FBD46D;
  --color-btn-active: #F76B8A;
  --checkbox-border: #4F8A8B;
  --checkbox-bg: #fff;
  --checkbox-checked-bg: #4F8A8B;
  --checkbox-checked-color: #fff;
  --history-bg: #fff;
  --history-border: #ccc;
  --history-thumb-bg: #FBD46D;
  --history-thumb-border: #4F8A8B;
  --history-thumb-hover: #F76B8A;
  --svg-border: #ccc;
  --svg-bg: #fff;
  --svg-shadow: 0 2px 8px rgba(0,0,0,0.08);
  --bg: #f4f8ff;
  --card: #ffffff;
  --ink: #0f1b3d;
  --muted:#6b7a99;
  --accent:#2157f2;
  --accent-2:#ff9d00;
  --ring:#cfe0ff;
  --border:#e7ecf6;
  --radius:22px;
  --shadow: 0 12px 35px rgba(24,51,116,.08);
  --shadow-strong: 0 16px 50px rgba(24,51,116,.14);
  --speed:180ms;
}

*{box-sizing:border-box;}
html,body{height:100%;}
body{
  margin:0;
  background: var(--bg);
```

```
  color: var(--ink);
  font-family:"Mulish",system-ui,-apple-system,Segoe
UI,Roboto,Helvetica,Arial,sans-serif;
  -webkit-font-smoothing:antialiased;
}

a{color:inherit;text-decoration:none;}

.wrap{
  width:min(1200px, 96%);
  margin: 42px auto;
  position:relative;
  display:flex;
  gap:24px;
}

.card{
  background: var(--card);
  border:1px solid var(--border);
  border-radius: var(--radius);
  box-shadow: var(--shadow);
}

.toolbar{
  flex-shrink:0;
  width:84px;
  padding:18px 12px;
  display:flex;
  flex-direction:column;
  align-items:center;
  gap:18px;
  position:sticky;
  top:24px;
  height:fit-content;
}

.logo{
  width:40px; height:40px;
  border-radius:14px;
  display:grid; place-items:center;
  background: linear-gradient(135deg, #2157f2, #3aa0ff);
  color:#fff; font-weight:800; font-size:18px;
}

.toolbar-nav{
  display:flex;
  flex-direction:column;
  gap:14px;
  width:100%;
}

.tool{
  display:flex;
  flex-direction:column;
```

```
  align-items:center;
  justify-content:center;
  gap:6px;
  padding:12px 6px;
  border-radius:16px;
  border:1px solid var(--border);
  background:#f6f9ff;
  color:#2f4c9c;
  transition:transform var(--speed), box-shadow var(--speed), background var(--
speed), color var(--speed);
  font-size:12px;
  text-align:center;
}
.tool:hover{ transform:translateY(-2px); box-shadow: var(--shadow); }
.tool.active{ background:#e9f1ff; color:#204399; }
.tool-icon{
  width:26px;
  height:26px;
  display:block;
}
.tool-label{ font-weight:600; font-size:11px; letter-spacing:0.02em; }

#main-panel{
  flex:1;
  padding:28px 26px;
  display:flex;
  flex-direction:column;
  gap:24px;
  margin-left:0;
  position:relative;
}

body.page-generator #main-panel{
  display:grid;
  grid-template-columns: minmax(0, 1fr) 420px;
  gap:26px;
}

.left{ padding:30px 22px 26px; }
.right{ padding:12px; }

.title{ font-size: clamp(26px, 3vw, 36px); margin: 0 0 6px; font-weight: 800;
color:#183056; }
.subtitle{ margin:0 0 18px; color: var(--muted); }

.auth-gate{
  padding:24px;
  border-radius:18px;
  border:1px dashed var(--border);
  background:#f6f9ff;
  color:var(--muted);
  display:grid;
  gap:12px;
}
```

```
.auth-gate .gate-actions{
  display:flex;
  gap:12px;
  flex-wrap:wrap;
}
.auth-gate .btn{
  width:fit-content;
}
#generatorFormWrap{
  display:grid;
  gap:18px;
}
.history-content{
  display:grid;
  gap:20px;
}
.profile-details{
  display:grid;
  gap:24px;
}

.input-card{
  background:#f8fbff;
  border:1px dashed #dfe8fb;
  border-radius: 18px;
  padding:14px;
  display:grid;
  gap:10px;
  transition: max-height 300ms ease, opacity 240ms ease, margin 240ms ease;
  overflow:hidden;
}
.input-card input{
  width:100%;
  padding:14px 14px;
  border:1px solid #dfe6f5;
  border-radius: 12px;
  outline:none;
  background:#fff;
  transition: box-shadow var(--speed), border var(--speed);
}
.input-card input:focus{
  border-color:#b8cdfc;
  box-shadow: 0 0 0 6px var(--ring);
}
.input-card .row{ display:flex; gap:10px; }

.btn{
  padding:12px 16px;
  border-radius: 12px;
  border:1px solid #d8e4ff;
  background:#eef4ff;
  color:#163a96;
  font-weight:700;
  cursor:pointer;
```

```
  transition: transform var(--speed), box-shadow var(--speed), background var(--
speed), filter var(--speed);
  display:inline-flex;
  align-items:center;
  justify-content:center;
  gap:8px;
}
.btn:hover{ transform: translateY(-1px); box-shadow: var(--shadow); }
.btn:active{ transform: translateY(0); }
.btn.primary{ background: var(--accent); color:#fff; border-color: transparent; }
.btn.accent{ background: var(--accent-2); color:#1a2652; border-color:
transparent; }
.btn.ghost{ background: rgba(33,87,242,.08); border-color: rgba(33,87,242,.12);
color:#1b3d8b; }
.btn.small{ padding:8px 10px; font-size: 12px; border-radius: 10px; }
.btn.danger{ background:#fee4e5; border-color:#facfd2; color:#aa1f2e; }
.btn.full{ width:100%; }

.preview{
  padding:18px;
  display:grid;
  gap:16px;
}

.qr-box {
  background: var(--card);
  color: var(--ink);
  border-radius: 20px;
  padding: 18px;
  min-height: 340px;
  display: grid;
  place-items: center;
  position: relative;
  box-shadow: var(--shadow-strong);
}
.qr-box img {
  max-width: 100%;
  max-height: 100%;
  display: none;
}
.qr-empty {
  color: #6b7a99;
  font-weight: 700;
  text-align: center;
  opacity: .8;
}

.controls{
  display:grid; gap:12px;
  background:#0d2f77;
  border-radius: 16px;
  padding:14px;
  color:#cfe0ff;
}
```

```
.control{ display:grid; gap:8px; }
.control label{ font-size:13px; color:#cfe0ff; }
.control-value{ font-size:11px; color:rgba(255,255,255,.7); margin-left:8px; font-
weight:600; }
.seg-2{ display:flex; gap:8px; align-items:center; }
.seg-2 input{ flex:1; padding:10px 12px; border-radius:10px; border:1px solid
#dfe6f5; }

.download-row{ display:flex; gap:12px; flex-wrap:wrap; }

.history-drawer{
  position: fixed;
  right: -420px; top: 0; bottom: 0;
  width: 420px; background: #ffffff; border-left:1px solid var(--border);
  box-shadow: var(--shadow-strong); padding: 16px; z-index: 30;
  transition: right 220ms ease;
}
.history-drawer.open{ right: 0; }
.drawer-head{ display:flex; align-items:center; justify-content:space-between;
margin-bottom: 10px; }
.history-list{ display:grid; gap:12px; max-height: calc(100vh - 120px);
overflow:auto; }
.history-card{
  border:1px solid var(--border);
  border-radius:14px;
  padding:12px;
  display:flex;
  gap:12px;
  align-items:center;
  background:#f9fbff;
}
.history-card img{ width:64px; height:64px; object-fit:contain; border-radius:8px;
background:#f5f7ff; }
.history-card-body{ flex:1; display:grid; gap:6px; min-width:0; }
.history-card-title{ font-weight:700; color:#1b2f59; overflow:hidden; text-
overflow:ellipsis; white-space:nowrap; }
.history-card-meta{ font-size:12px; color: var(--muted); }
.history-card-actions{ display:flex; gap:8px; flex-wrap:wrap; }

.toast{
  position: fixed; left:50%; bottom:22px; transform:translateX(-50%)
translateY(120%);
  background:#0b2a6a; color:#fff; padding:12px 16px; border-radius:14px;
  box-shadow: var(--shadow-strong); transition: transform var(--speed); z-
index:10;
}
.toast.show{ transform:translateX(-50%) translateY(0); }

.bg-shape{ position: fixed; border-radius: 50%; filter: blur(0.4px); opacity:.8;
z-index:-1; }
.s1{ width:520px; height:520px; background:#0e2b84; left:-160px; top:-120px; }
.s2{ width:160px; height:160px; background:#2aa6ff; right:80px; top:30%; }
.s3{ width:240px; height:240px; background:#ff9d00; right:-60px; bottom:8%; }
```

```
.home-hero{
  padding:48px;
  border-radius:20px;
  background:linear-gradient(135deg, rgba(33,87,242,.09), rgba(18,38,94,.18));
  display:grid;
  gap:18px;
  text-align:left;
}
.home-hero .hero-actions{ display:flex; gap:14px; flex-wrap:wrap; }

.auth-panel{
  display:grid;
  gap:24px;
  grid-template-columns: minmax(0, 1fr) 360px;
  align-items:start;
}
.auth-card{
  background:#f8fbff;
  border-radius:18px;
  border:1px solid #dfe6f5;
  padding:24px;
  display:grid;
  gap:16px;
}
.form-field{ display:grid; gap:8px; font-weight:600; color: var(--muted); }
.form-field input{
  padding:14px 14px;
  border-radius:12px;
  border:1px solid #dfe6f5;
  background:#fff;
}
.form-field input:focus{
  border-color:#b8cdfc;
  box-shadow:0 0 0 6px var(--ring);
}
.auth-link{ font-size:14px; color: var(--muted); text-align:center; }
.auth-link a{ color: var(--accent); font-weight:700; }

.profile-panel{ display:grid; gap:24px; }
.profile-card{
  padding:24px;
  border:1px solid var(--border);
  border-radius:18px;
  background:#f9fbff;
  display:grid;
  gap:18px;
}
.profile-card h2{ margin:0; font-size:20px; }
.profile-info{ display:grid; gap:12px; margin:0; }
.profile-info div{ display:flex; justify-content:space-between; align-
items:center; }
.profile-info dt{ font-weight:700; color:#1c2f57; }
.profile-info dd{ margin:0; color: var(--muted); }
.profile-actions{ display:flex; gap:12px; flex-wrap:wrap; }
```

```
.profile-form{ display:grid; gap:14px; }

.history-panel{ display:grid; gap:20px; }
.history-actions{ display:flex; gap:12px; flex-wrap:wrap; align-items:center; }
.history-grid{ display:grid; gap:16px; }
.history-grid.empty{ opacity:0.7; }
.history-empty{ padding:24px; border:1px dashed var(--border); border-radius:16px;
background:#f6f9ff; color: var(--muted); }

.hidden{ display:none !important; }

@media (max-width: 1080px){
  .wrap{ flex-direction:column; }
  .toolbar{ flex-direction:row; width:100%; position:static; justify-
content:space-between; }
  .toolbar-nav{ flex-direction:row; justify-content:center; }
  .tool{ flex:1; }
  #main-panel{ margin:0; }
  body.page-generator #main-panel{ grid-template-columns: 1fr; }
  .auth-panel{ grid-template-columns: 1fr; }
}

@media (max-width: 720px){
  #main-panel{ padding:20px 18px; }
  .home-hero{ padding:32px; }
  .history-card{ flex-direction:column; align-items:flex-start; }
  .history-card-actions{ width:100%; }
}

.visually-hidden{ position:absolute; left:-9999px; width:1px; height:1px;
overflow:hidden; }


.input-card{
  background:#f8fbff;
  border:1px dashed #dfe8fb;
  border-radius: 18px;
  padding:14px;
  display:grid;
  gap:10px;
  overflow:hidden;
}
#saveQr{
  margin-top:14px;
  width:fit-content;
}
#saveQr:disabled{
  opacity:0.45;
  cursor:not-allowed;
}

/* Cleaned up .checkbox-inline rule */
.checkbox-inline {
  display: flex;
```

```
  align-items: center;
  gap: 6px;
  font-size: 12px;
  color: #cfe0ff;
}
.checkbox-inline input {
  width: auto;
  height: auto;
}
.profile-form input[disabled]{
  background:#eef2f9;
  color:var(--muted);
  cursor:not-allowed;
}
```

Root

storage.py

```
```

templates

base.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>{% block title %}QR Forge{% endblock %}</title>
  <meta name="viewport" content="width=device-width, initial-scale=1" />

  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link href="https://fonts.googleapis.com/css2?
family=Mulish:ital,wght@0,200..1000;1,200..1000&display=swap" rel="stylesheet">

  <link rel="stylesheet" href="/static/styles.css" />
  {% block head_scripts %}{% endblock %}
</head>
<body class="page-{{ active_page|default('home') }}">
  <div class="bg-shape s1"></div>
  <div class="bg-shape s2"></div>
  <div class="bg-shape s3"></div>

  <div class="wrap">
    <aside class="toolbar card">
```

```
      <div class="logo">QF</div>
      <nav class="toolbar-nav">
        <a id="navHome" href="{{ url_for('home') }}" class="tool{% if active_page
== 'home' %} active{% endif %}" title="Home">
          <img src="/assets/icons/home.svg" alt="" class="tool-icon" />
          <span class="tool-label">Home</span>
        </a>
        <a id="navGenerator" href="{{ url_for('generator_page') }}" class="tool{%
if active_page == 'generator' %} active{% endif %}" title="Generator">
          <img src="/assets/icons/generator.svg" alt="" class="tool-icon" />
          <span class="tool-label">Generator</span>
        </a>
        <a id="navHistory" href="{{ url_for('history_page') }}" class="tool{% if
active_page == 'history' %} active{% endif %}" title="History">
          <img src="/assets/icons/history.svg" alt="" class="tool-icon" />
          <span class="tool-label">History</span>
        </a>
        <a id="navProfile" href="{{ url_for('profile_page') }}" class="tool{% if
active_page == 'profile' %} active{% endif %}" title="Profile" data-requires-auth>
          <img src="/assets/icons/profile.svg" alt="" class="tool-icon" />
          <span class="tool-label">Profile</span>
        </a>
        <a id="navLogin" href="{{ url_for('login_page') }}" class="tool{% if
active_page == 'login' %} active{% endif %}" title="Login" data-hide-when-authed>
          <img src="/assets/icons/login.svg" alt="" class="tool-icon" />
          <span class="tool-label">Login</span>
        </a>
      </nav>
    </aside>

    <main class="panel card" id="main-panel">
      {% block content %}{% endblock %}
    </main>
  </div>

  <div id="toast" class="toast" role="status" aria-live="polite"></div>
  {% block body_scripts %}{% endblock %}
  <script defer src="/static/app.js"></script>
</body>
</html>
```

history.html

```
{% extends "base.html" %}
{% block title %}History · QR Forge{% endblock %}

{% block content %}
<section class="history-panel">
  <div id="historyGuard" class="auth-gate">
    <h2>Sign in to view your QR history</h2>
```

```
    <p>Login to retrieve saved QR codes, download exports, and manage your
library.</p>
    <div class="gate-actions">
      <a class="btn primary" href="{{ url_for('login_page') }}">Login</a>
      <a class="btn ghost" href="{{ url_for('signup_page') }}">Create account</a>
    </div>
  </div>

  <div id="historyContent" class="history-content" data-history-auth>
    <header class="history-header">
      <h1 class="title">QR history</h1>
      <p class="subtitle">All of your generated QR codes, ready to download again
or remove.</p>
    </header>

    <div class="history-actions">
      <button id="refreshHistory" class="btn ghost" type="button">Refresh</button>
      <button id="exportCsv" class="btn" type="button">Export CSV</button>
    </div>

    <div id="historyEmpty" class="history-empty hidden">No QR codes yet. Generate
your first one from the generator tab.</div>
    <div id="historyGrid" class="history-grid"></div>
  </div>
</section>
{% endblock %}
```

home.html

```
{% extends "base.html" %}
{% block title %}Home · QR Forge{% endblock %}

{% block content %}
<section class="home-hero">
  <h1 class="title">Design. Personalize. Share.</h1>
  <p class="subtitle">QR Forge lets you craft branded QR codes with color,
padding, and overlays in seconds.</p>
  <div class="hero-actions">
    <a class="btn primary" href="{{ url_for('generator_page') }}">Start
generating</a>
    <a class="btn ghost" href="{{ url_for('history_page') }}">View history</a>
  </div>
</section>
{% endblock %}
```

index.html

```
{% extends "base.html" %}
{% block title %}Generator · QR Forge{% endblock %}

{% block content %}
<section class="left">
  <div id="generatorGuard" class="auth-gate">
    <h2>Sign in to generate QR codes</h2>
    <p>Create an account to store your designs, customize styles, and download SVG
or PNG exports.</p>
    <div class="gate-actions">
      <a class="btn primary" href="{{ url_for('login_page') }}">Login</a>
      <a class="btn ghost" href="{{ url_for('signup_page') }}">Create account</a>
    </div>
  </div>

  <div id="generatorFormWrap" data-generator-auth>
    <h1 class="title">Enter your text</h1>
    <p class="subtitle">Preview your QR code and save it when you're ready.</p>

    <form id="qr-form" class="input-card">
      <input id="url" name="url" type="url" placeholder="https://example.com"
required />
      <div class="row">
        <input id="title" name="title" type="text" placeholder="QR title
(optional)" />
        <button id="previewBtn" type="submit" class="btn primary">Preview</button>
      </div>
      <div class="controls" id="controls">
        <div class="control">
          <label>Foreground color</label>
          <input id="fgColor" type="color" value="#0a0a0a" />
        </div>
        <div class="control">
          <label>Background color</label>
          <div class="seg-2">
            <input id="bgColor" type="color" value="#ffffff" />
            <label class="checkbox-inline">
              <input id="bgTransparent" type="checkbox" /> Transparent
            </label>
          </div>
        </div>
        <div class="control">
          <label>Size <span id="sizeValue" class="control-value">512 px</span>
</label>
          <input id="sizeRange" type="range" min="128" max="1024" step="64"
value="512" />
        </div>
        <div class="control">
          <label>Padding <span id="paddingValue" class="control-value">16
px</span></label>
          <input id="paddingRange" type="range" min="0" max="80" step="4"
value="16" />
        </div>
```

```
          <div class="control">
            <label>Border radius</label>
            <input id="radiusRange" type="range" min="0" max="60" value="20" />
          </div>
        </div>
        <button id="saveQr" class="btn" type="button" disabled>Save to
history</button>
      </form>
    </div>
</section>

<section class="right" data-generator-auth>
  <div class="preview card">
    <div class="qr-box" id="qrBox">
      <img id="qrPreview" alt="QR preview" />
      <div id="qrEmpty" class="qr-empty">Your QR preview will appear here</div>
    </div>
    <div class="download-row">
      <button id="dlSvg" class="btn" type="button">Download SVG</button>
      <button id="dlPng" class="btn accent" type="button">Download PNG</button>
      <button id="openHistory" class="btn ghost" type="button">History</button>
    </div>
  </div>
</section>

<aside id="historyDrawer" class="history-drawer" data-generator-auth>
  <div class="drawer-head">
    <h3>Recent QRs</h3>
    <button id="closeHistory" class="btn ghost small" type="button">Close</button>
  </div>
  <div id="historyList" class="history-list"></div>
</aside>
{% endblock %}
```

login.html

```
{% extends "base.html" %}
{% block title %}Login · QR Forge{% endblock %}

{% block content %}
<section class="auth-panel">
  <div class="auth-copy">
    <h1 class="title">Welcome back</h1>
    <p class="subtitle">Sign in to manage your QR history and profile.</p>
  </div>
  <form id="login-form" class="auth-card">
    <label class="form-field">
      <span>Email</span>
      <input type="email" name="email" placeholder="you@example.com" required />
    </label>
```

```
      <label class="form-field">
        <span>Password</span>
        <input type="password" name="password" placeholder="••••••••" required />
      </label>
      <button type="submit" class="btn primary full">Login</button>
      <p class="auth-link">
        No account yet?
        <a href="{{ url_for('signup_page') }}">Create one</a>
      </p>
    </form>
  </section>
  {% endblock %}
```

profile.html

```
{% extends "base.html" %}
{% block title %}Profile · QR Forge{% endblock %}

{% block content %}
<section class="profile-panel">
  <div id="profileGuard" class="auth-gate">
    <h2>You are not signed in</h2>
    <p>Login to edit your profile, review account details, and manage security
settings.</p>
    <div class="gate-actions">
      <a class="btn primary" href="{{ url_for('login_page') }}">Login</a>
      <a class="btn ghost" href="{{ url_for('signup_page') }}">Create account</a>
    </div>
  </div>

  <div id="profileContent" class="profile-details hidden" data-profile-auth>
    <header class="profile-header">
      <h1 class="title">Your profile</h1>
      <p class="subtitle">Manage your personal details and security preferences.
</p>
    </header>

    <section class="profile-card">
      <h2>Account</h2>
      <form id="profileForm" class="profile-form">
        <label class="form-field">
          <span>Full name</span>
          <input id="profileFullName" type="text" name="full_name"
placeholder="Your name" />
        </label>
        <label class="form-field">
          <span>Email</span>
          <input id="profileEmail" type="email" disabled />
        </label>
        <label class="form-field">
```

```
          <span>Member since</span>
          <input id="profileSince" type="text" disabled />
        </label>
        <label class="form-field">
          <span>New password (optional)</span>
          <input id="profilePassword" type="password" name="password"
placeholder="••••••••" />
        </label>
        <div class="profile-actions">
          <button class="btn primary" id="saveProfile" type="submit">Save
changes</button>
          <button class="btn" id="logoutBtn" type="button">Logout</button>
          <button class="btn danger" id="deleteAccount" type="button">Delete
account</button>
        </div>
      </form>
    </section>

    <section class="profile-card">
      <h2>Security</h2>
      <p class="subtitle">Keep your account safe by using a strong password.
Password updates are currently handled above.</p>
    </section>
  </div>
</section>
{% endblock %}
```

signup.html

```
{% extends "base.html" %}
{% block title %}Sign Up · QR Forge{% endblock %}

{% block content %}
<section class="auth-panel">
  <div class="auth-copy">
    <h1 class="title">Create your account</h1>
    <p class="subtitle">Track your QR creations across devices.</p>
  </div>
  <form id="signup-form" class="auth-card">
    <label class="form-field">
      <span>Full name</span>
      <input type="text" name="full_name" placeholder="Ada Lovelace" required />
    </label>
    <label class="form-field">
      <span>Email</span>
      <input type="email" name="email" placeholder="you@example.com" required />
    </label>
    <label class="form-field">
      <span>Password</span>
      <input type="password" name="password" placeholder="At least 8 characters"
```

```
required />
    </label>
    <button type="submit" class="btn primary full">Create account</button>
    <p class="auth-link">
      Already have an account?
      <a href="{{ url_for('login_page') }}">Login</a>
    </p>
  </form>
</section>
{% endblock %}
```

tests

conftest.py

```
import sys
from pathlib import Path
from typing import Generator

ROOT_DIR = Path(__file__).resolve().parents[1]
if str(ROOT_DIR) not in sys.path:
    sys.path.append(str(ROOT_DIR))

import pytest
from fastapi.testclient import TestClient
from sqlalchemy.pool import StaticPool
from sqlmodel import SQLModel, Session, create_engine

from app import app
from db import get_session

TEST_DATABASE_URL = "sqlite://"


def _create_engine():
    return create_engine(
        TEST_DATABASE_URL,
        connect_args={"check_same_thread": False},
        poolclass=StaticPool,
    )


@pytest.fixture(scope="session")
def engine() -> Generator:
    engine = _create_engine()
    yield engine


@pytest.fixture(autouse=True)
def prepare_database(engine) -> Generator:
```

```
    SQLModel.metadata.drop_all(engine)
    SQLModel.metadata.create_all(engine)
    yield
    SQLModel.metadata.drop_all(engine)


@pytest.fixture()
def client(tmp_path: Path, monkeypatch, engine) -> TestClient:
    def override_get_session() -> Generator[Session, None, None]:
        with Session(engine) as session:
            yield session

    app.dependency_overrides[get_session] = override_get_session

    from routers import qr

    svg_dir = tmp_path / "svg"
    png_dir = tmp_path / "png"
    svg_dir.mkdir(parents=True, exist_ok=True)
    png_dir.mkdir(parents=True, exist_ok=True)
    monkeypatch.setattr(qr, "SVG_DIR", svg_dir, raising=False)
    monkeypatch.setattr(qr, "PNG_DIR", png_dir, raising=False)

    return TestClient(app)
```

test_auth.py

```
from fastapi.testclient import TestClient


def register_user(client: TestClient, email: str, password: str) -> None:
    payload = {
        "email": email,
        "full_name": "Test User",
        "password": password,
    }
    resp = client.post("/api/auth/signup", json=payload)
    assert resp.status_code == 201, resp.text


def authenticate(client: TestClient, email: str, password: str) -> dict:
    register_user(client, email, password)
    resp = client.post(
        "/api/auth/login",
        json={"email": email, "password": password},
    )
    assert resp.status_code == 200, resp.text
    return {"Authorization": f"Bearer {resp.json()['access_token']}"}
```

```python
def test_signup_duplicate_email(client: TestClient) -> None:
    register_user(client, "dup@example.com", "password123")
    resp = client.post(
        "/api/auth/signup",
        json={
            "email": "dup@example.com",
            "full_name": "Another",
            "password": "password123",
        },
    )
    assert resp.status_code == 409
    assert "Email already registered" in resp.text


def test_login_invalid_password(client: TestClient) -> None:
    register_user(client, "bob@example.com", "password123")
    resp = client.post(
        "/api/auth/login",
        json={"email": "bob@example.com", "password": "wrongpass"},
    )
    assert resp.status_code == 401
    assert "Invalid credentials" in resp.text


def test_logout_endpoint(client: TestClient) -> None:
    headers = authenticate(client, "logout@example.com", "password123")
    resp = client.post("/api/auth/logout", headers=headers)
    assert resp.status_code == 200
    assert resp.json()["ok"] is True


def test_login_missing_user(client: TestClient) -> None:
    resp = client.post(
        "/api/auth/login",
        json={"email": "ghost@example.com", "password": "password123"},
    )
    assert resp.status_code == 401
```

test_auth_and_qr.py

```python
import base64

import pytest
from fastapi.testclient import TestClient
from sqlmodel import Session, select

from models import QRItem, User


def _auth_headers(client: TestClient, email: str = "alice@example.com", password:
```

```
str = "secret123") -> dict:
    signup_payload = {
        "email": email,
        "full_name": "Alice Example",
        "password": password,
    }
    resp = client.post("/api/auth/signup", json=signup_payload)
    assert resp.status_code == 201, resp.text

    login_payload = {"email": email, "password": password}
    resp = client.post("/api/auth/login", json=login_payload)
    assert resp.status_code == 200, resp.text
    token = resp.json()["access_token"]
    return {"Authorization": f"Bearer {token}"}


def test_create_qr_requires_auth(client: TestClient) -> None:
    resp = client.post(
        "/api/qr",
        json={"title": "Unauth", "url": "https://example.com"},
    )
    assert resp.status_code == 401


def test_preview_requires_auth(client: TestClient) -> None:
    resp = client.post(
        "/api/qr/preview",
        json={
            "title": "No auth",
            "url": "https://example.com",
            "foreground_color": "#000000",
            "background_color": "#ffffff",
            "size": 256,
            "padding": 8,
            "border_radius": 8,
        },
    )
    assert resp.status_code == 401


def test_preview_and_lifecycle(client: TestClient) -> None:
    headers = _auth_headers(client)

    payload = {
        "title": "My QR",
        "url": "https://example.com",
        "foreground_color": "#123456",
        "background_color": "transparent",
        "size": 320,
        "padding": 12,
        "border_radius": 24,
    }
    preview_resp = client.post("/api/qr/preview", json=payload, headers=headers)
    assert preview_resp.status_code == 200, preview_resp.text
```

```
    preview = preview_resp.json()
    assert preview["png_data"]
    base64.b64decode(preview["png_data"])

    create_resp = client.post("/api/qr", json=payload, headers=headers)
    assert create_resp.status_code == 201, create_resp.text
    created = create_resp.json()
    assert created["background_color"] == "transparent"

    list_resp = client.get("/api/qr", headers=headers)
    assert list_resp.status_code == 200
    items = list_resp.json()
    assert len(items) == 1

    download_png = client.get(
        f"/api/qr/{created['id']}/download",
        params={"format": "png"},
        headers=headers,
    )
    assert download_png.status_code == 200
    assert download_png.headers["content-type"] == "image/png"

    delete_resp = client.delete(f"/api/qr/{created['id']}", headers=headers)
    assert delete_resp.status_code == 200
    assert delete_resp.json()["ok"] is True

    empty_resp = client.get("/api/qr", headers=headers)
    assert empty_resp.status_code == 200
    assert empty_resp.json() == []


def test_delete_user_removes_qrs(client: TestClient, engine) -> None:
    headers = _auth_headers(client)
    payload = {
        "title": "Keep",
        "url": "https://example.com",
        "foreground_color": "#000000",
        "background_color": "#ffffff",
        "size": 256,
        "padding": 8,
        "border_radius": 12,
    }
    resp = client.post("/api/qr", json=payload, headers=headers)
    assert resp.status_code == 201

    del_resp = client.delete("/api/user/me", headers=headers)
    assert del_resp.status_code == 200
    assert del_resp.json()["ok"] is True

    with Session(engine) as session:
        assert session.exec(select(User)).first() is None
        assert session.exec(select(QRItem)).all() == []
```

```
def test_update_profile_and_password_flow(client: TestClient) -> None:
    headers = _auth_headers(client, email="bob@example.com",
password="initial123")

    update_resp = client.patch(
        "/api/user/me",
        json={"full_name": "Bob Builder", "password": "newsecret456"},
        headers=headers,
    )
    assert update_resp.status_code == 200, update_resp.text
    updated = update_resp.json()
    assert updated["full_name"] == "Bob Builder"

    me_resp = client.get("/api/user/me", headers=headers)
    assert me_resp.status_code == 200
    assert me_resp.json()["full_name"] == "Bob Builder"

    old_login = client.post(
        "/api/auth/login",
        json={"email": "bob@example.com", "password": "initial123"},
    )
    assert old_login.status_code == 401

    new_login = client.post(
        "/api/auth/login",
        json={"email": "bob@example.com", "password": "newsecret456"},
    )
    assert new_login.status_code == 200
    assert "access_token" in new_login.json()
```

test_qr.py

```
from fastapi.testclient import TestClient

def auth_headers(client: TestClient, email: str = "qrtester@example.com") -> dict:
    payload = {
        "email": email,
        "full_name": "QR Tester",
        "password": "strongpass123",
    }
    resp = client.post("/api/auth/signup", json=payload)
    assert resp.status_code == 201
    resp = client.post(
        "/api/auth/login",
        json={"email": payload["email"], "password": payload["password"]},
    )
    assert resp.status_code == 200
    return {"Authorization": f"Bearer {resp.json()['access_token']}"}
```

```python
def test_create_qr_invalid_payload(client: TestClient) -> None:
    headers = auth_headers(client)
    resp = client.post(
        "/api/qr",
        json={
            "title": "Bad QR",
            "url": "not-a-url",
            "foreground_color": "#000000",
            "background_color": "#ffffff",
            "size": 320,
            "padding": -1,
            "border_radius": 10,
        },
        headers=headers,
    )
    assert resp.status_code == 422


def test_download_requires_authentication(client: TestClient) -> None:
    headers = auth_headers(client)
    payload = {
        "title": "Private",
        "url": "https://example.com",
        "foreground_color": "#000000",
        "background_color": "#ffffff",
        "size": 256,
        "padding": 8,
        "border_radius": 8,
    }
    create_resp = client.post("/api/qr", json=payload, headers=headers)
    assert create_resp.status_code == 201
    qr_id = create_resp.json()["id"]

    resp = client.get(f"/api/qr/{qr_id}/download")
    assert resp.status_code == 401


def test_history_only_returns_owner_items(client: TestClient) -> None:
    alice_headers = auth_headers(client)
    client.post(
        "/api/qr",
        json={
            "title": "Alice QR",
            "url": "https://alice.example.com",
            "foreground_color": "#123123",
            "background_color": "#ffffff",
            "size": 256,
            "padding": 8,
            "border_radius": 8,
        },
        headers=alice_headers,
    )

    # create second user
```

```
    headers_bob = auth_headers(client, email="bob@example.com")
    resp = client.get("/api/qr/history", headers=headers_bob)
    assert resp.status_code == 200
    assert resp.json() == []

    resp_alice = client.get("/api/qr/history", headers=alice_headers)
    assert len(resp_alice.json()) == 1
```