**1. Core Backend Language & Framework**

Since you already picked **Rust**:

* **Rust language mastery** → Ownership, Borrowing, Traits, Async (tokio), Error Handling.
* **Backend framework** →
  + **Axum** (clean, modern, async-friendly) or
  + **Actix-web** (high performance, mature).

**2. Databases**

Real estate + school systems rely heavily on structured data.

* **PostgreSQL** → For relational data (properties, tenants, contracts, school data).
* **SQLx** or **Diesel ORM** → For Rust database interaction.
* Learn **migrations**, indexing, and query optimization.

**3. Authentication & Authorization**

You’ll need **secure logins** for tenants, landlords, teachers, students, admins.

* JWT authentication in Rust.
* Role-based access control (RBAC).

**4. API Design**

Both MVPs will need **REST APIs** (and maybe GraphQL later).

* Learn **API versioning**, request validation, pagination, error handling.
* Use **OpenAPI/Swagger** for documentation.

**5. File Handling & Media Storage**

For real estate → property images, contracts; for Learnza → documents, timetables.

* Store files in **AWS S3**, **Supabase storage**, or **Google Cloud Storage**.

**6. Real-Time Communication**

For things like live chat between agents and tenants, or between teachers and students.

* **WebSockets** in Rust (via tokio-tungstenite).

**7. AI/ML Integration**

For Learnza’s exam predictions & timetable optimization:

* **Python** (pandas, scikit-learn, FastAPI for ML APIs).
* Deploy AI models and connect them to Rust backend via HTTP or gRPC.

**8. Deployment & DevOps**

You’ll need to run your MVP somewhere.

* **Docker** → containerize your backend.
* **Railway.app / Render / Fly.io** for early deployment.
* **CI/CD** → GitHub Actions to automate deployment.

**9. Optional but Useful**

* **Redis** → for caching (faster property search, timetable queries).
* **ElasticSearch** → for advanced property search filters.