Rapid Prototyping in TypeScript

Studienarbeit

Students

Silvan Kisseleff

Isaia Brassel

Initial Situation: Building a fullstack web application with NodeJS in today's landscape of countless frameworks and libraries can be daunting. Which one should you choose? How do you configure the chosen ones to work together seamlessly rather than against each other? The available templates tend to be overloaded with boilerplate code, which slows down the development process and distracts users from focusing on what truly matters: building features.

Approach / Technology: Research into rapid prototyping tools highlighted the potential to significantly simplify application development by focusing on adaptability and efficiency. To achieve this, we selected tools and techniques to build an application with a GraphQL API that dynamically adjusts through generated code. This setup eliminates the need for manual updates whenever changes are introduced, making the process faster and more reliable.

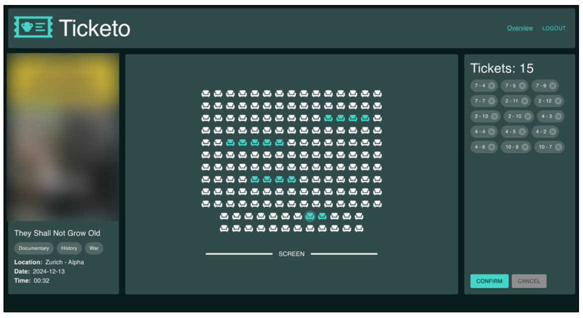
To help users understand the dynamic approach, we created a demo application and a tutorial. The demo application is an online ticket reservation for a cinema. We chose this domain because it is easy to understand. The tutorial guides users through every aspect of our rapid prototyping method in an interactive way. As users work through the tutorial, they gain practical experience with the tools and techniques, enabling them to apply this approach to their own projects.

The GraphQL API is implemented with Apollo Server, combined with automatically generated resolvers based on TypeGraphQL-Prisma. TypeGraphQL-Prisma provides out-of-the-box support for create, read, update, and delete (CRUD) operations for every data model, reducing the effort required to build and maintain the API. For custom operations we leverage TypeGraphQL so that complex processes can still be realized. The frontend is built with React. This setup ensures a dynamic, responsive user experience. For the tutorial, we used Docusaurus, a static website generator that makes it easy to build documentation-focused web applications that are accessible and engaging for users to learn and explore our approach.

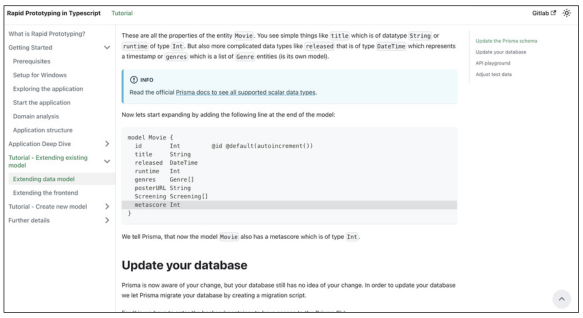
This combination of technologies streamlines the development process, providing both a practical introduction to rapid prototyping and a robust foundation for scalable application development.

Result: By writing an easy-to-understand tutorial and creating a demo application in a well-known domain, we created a product that brings rapid prototyping to the developer. The tutorial shows how to adapt the already existing features with rapid prototyping. It is also possible to create new features from scratch. To assure that the user does not get stuck, we provide a solution for every single step, which users can look at if they want to. We also ensured that the user does not have to waste time on a complicated setup by implementing the entire application in Docker containers. Different user tests showed that the tutorials indeed are easy to understand and intuitive and that our template increases productivity.

Ticket reservation application in which available seats can be selected
Own presentment



Tutorial page explaining how to update the database
Own presentment



These are all the properties of the entity `Movie`. You see simple things like `title` which is of datatype `String` or `runtime` of type `Int`. But also more complicated data types like `released` that is of type `DateTime` which represents a timestamp or `genres` which is a list of `Genre` entities (is its own model).

INFO
Read the official Prisma docs to see all supported scalar data types.

Now lets start expanding by adding the following line at the end of the model:

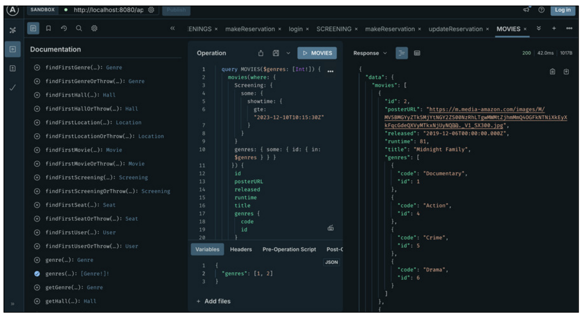
```
model Movie {
  id        Int         @id @default(autoincrement())
  title     String
  released  DateTime
  runtime   Int
  genres    Genre[]
  posterURL String
  Screening Screening[]
  metascore Int
}
```

We tell Prisma, that now the model `Movie` also has a metascore which is of type `Int`.

Update your database

Prisma is now aware of your change, but your database still has no idea of your change. In order to update your database we let Prisma migrate your database by creating a migration script.

GraphQL API playground demonstrating a sample request
Own presentment



```
findFirstGenre(…): Genre
findFirstGenreOrThrow(…): Genre
findFirstHall(…): Hall
findFirstHallOrThrow(…): Hall
findFirstLocation(…): Location
findFirstLocationOrThrow(…): Location
findFirstMovie(…): Movie
findFirstMovieOrThrow(…): Movie
findFirstScreening(…): Screening
findFirstScreeningOrThrow(…): Screening
findFirstSeat(…): Seat
findFirstSeatOrThrow(…): Seat
findFirstUser(…): User
findFirstUserOrThrow(…): User
genre(…): Genre
genres(…): [Genre!]!
getGenre(…): Genre
getHall(…): Hall
```

```
query MOVIES($genres: [Int!]) {
  movies(where: {
    Screening: {
      some: {
        showtime: {
          gte:
          "2023-12-10T10:15:30Z"
        }
      }
    }
    genres: { some: { id: { in:
    $genres } } }
  }) {
    id
    posterURL
    released
    runtime
    title
    genres {
      code
      id
    }
```

```
{
  "genres": [1, 2]
}
```

```
{
  "data": {
    "movies": [
      {
        "id": 2,
        "posterURL": "https://m.media-amazon.com/images/M/MV5BMGYyZTk5MjYtNGY2ZS00NzRhLTgwMWMtZjhmMmQ4OGFkNTNiXkEyXkFqcGdeQXVyMTkxNjUyNQ@@._V1_SX300.jpg",
        "released": "2019-12-06T00:00:00.000Z",
        "runtime": 81,
        "title": "Midnight Family",
        "genres": [
          {
            "code": "Documentary",
            "id": 1
          },
          {
            "code": "Action",
            "id": 4
          },
          {
            "code": "Crime",
            "id": 5
          },
          {
            "code": "Drama",
            "id": 6
          }
        ]
      },
      {
```

Advisor
Prof. Dr. Olaf Zimmermann

Co-Examiner
Prof. Dr. Olaf Zimmermann, Rapperswil, SG

Subject Area
Application Design, Software