GraphQL Cheatsheet:

Client:

Einfache Query
```
query getProducers {
   producers(name:"Peter") {
       id
       username
       products(name: "Apfel") {
           id
           name
       }
   }
}
```

Einfache Query

```
query getProducers {
   producers(name:”Peter”) {
       id
       username
       products(name: “Apfel”) {
           id
           name
       }
   }
}
```

Mutation mit Objekt als Input (und einem Enum-Wert):

```
mutation createProduct {
 createProduct(
   producerId: "d467f50a"
   productInput: {
     name: "Banane"
     unit: KILOGRAM
     price_per_unit: 2.50
   }
 ) {
   id
   name
 }
}
```

Variablen Definition:

```
{
 "minimum_rating": 3
}
```

Variablen Gebrauch:

```
query getProducers($minimum_rating: Float!) {
 producers(rating: $minimum_rating) {
   id
 }
}
```

Fragment Definition:

```
fragment UserFragment on User {
 id
 username
 email
}
```

Fragment Gebrauch:

```
query getUsers {
 users {
    ...UserFragment
 }
}
```

Inline-Fragment Gebrauch:

```
transfer_accounts {
    ... on Paypal {
      email
    }
    ... on Bank {
      account_number
      bank_code
      bank_name
    }
}
```

Server:

Object-Type Definition:

```graphql
type User {
   id: ID!
   username: String!
   email: String!
}
```

Query-Type Definition:

```graphql
type Query {
    users : [User!]
    user(id: ID!) : User
}
```

Input-Type Definition:

```graphql
input UserUpdateInput {
    name: String
    email: String
}
```

Mutation-Type Definition (mit Input-Type):

```graphql
type Mutation {
      updateUser(id: ID!, input: UserUpdateInput) : User
}
```

Query / Mutation-Resolver:

```
const userDB = require(“../database/user.db”)

module.exports = {
    Query: {
        user: (parent, args, context, info) => {
            const { id } = args // const id = args.id
            return userDB.getUserById(id)
        }
    },
    Mutation: {
        ...
    }
}
```

Object- / Attribute-Resolver:

```
const companyDB = require(“../database/company.db”)

module.exports = {
   User: {
        company: (parent, args, context, info) => {
            const { companyId } = parent
            return companyDB.getCompanyById(companyId)
        }
    }
}
```

Enum-Type Definieren:

```
enum Day {
    MONDAY
    TUESDAY
    WEDNESDAY
    THURSDAY
    FRIDAY
    SATURDAY
    SUNDAY
}
```

Enum-Resolver:

```
module.exports = {
    Day: {
        MONDAY: "monday",
        TUESDAY: "tuesday",
        WEDNESDAY: "wednesday",
        THURSDAY: "thursday",
        FRIDAY: "friday",
        SATURDAY: "saturday",
        SUNDAY: "sunday"
    }
}
```

Interface & Sub-Typen Definieren:

```
interface User {
    id: ID!
    username: String!
    email: String!
}
```

```
type Consumer implements User {
   id: ID!
   username: String!
   email: String!
   purchases: [Product!]
}

type Producer implements User {
   id: ID!
   username: String!
   email: String!
   business_days: [Day!]!
   company: Company
   products: [Product!]
}
```

Union-Type Definieren:

```
union TransferAccount = Paypal | Bank

type Paypal {
   email: String!
}

type Bank {
   account_number: String!
   bank_code: String!
   bank_name: String!
}
```

__resolveType Resolver:

```
module.exports = {
    User: {
        __resolveType: (user) => {
            switch(user.type) {
                case “producer”: “Producer”,
                case “consumer”: “Consumer”,
                default: throw Error(“Could not identify”)
            }
        }
    },
}
```

Subscription-Type Definieren:

```
type Subscription {
   reviewAdded(producerId: ID!) : Review!
}
```

Einfacher Subscription Resolver (Änderungen an allen Objekten Abonnieren):

```
const { GraphQLServer, PubSub } = require('graphql-yoga') //apollo-server
const pubsub = new PubSub()

Subscription: {
    reviewAdded: {
        subscribe: (_parent, _args, _context, _info) => {
            return pubsub.asyncIterator(“Review_Added_Channel”)
        },
    },
},
```

Subscription Resolver Mit Filtern:

```
const { GraphQLServer, PubSub } = require('graphql-yoga') //apollo-server
const pubsub = new PubSub()

Subscription: {
    reviewAdded: {
        subscribe: withFilter(
            (_parent, _args, _context, _info) => {
                return pubsub.asyncIterator(“Review_Added_Channel”)
            },
            (payload, variables) =>
                payload.reviewAdded.producerId === variables.producerId
        )
    },
},
```

Subscription Abschicken / Anstoßen:

```
const { GraphQLServer, PubSub } = require('graphql-yoga') //apollo-server
const pubsub = new PubSub()

pubsub.publish(“Review_Added_Channel”, { reviewAdded: newReview })
```