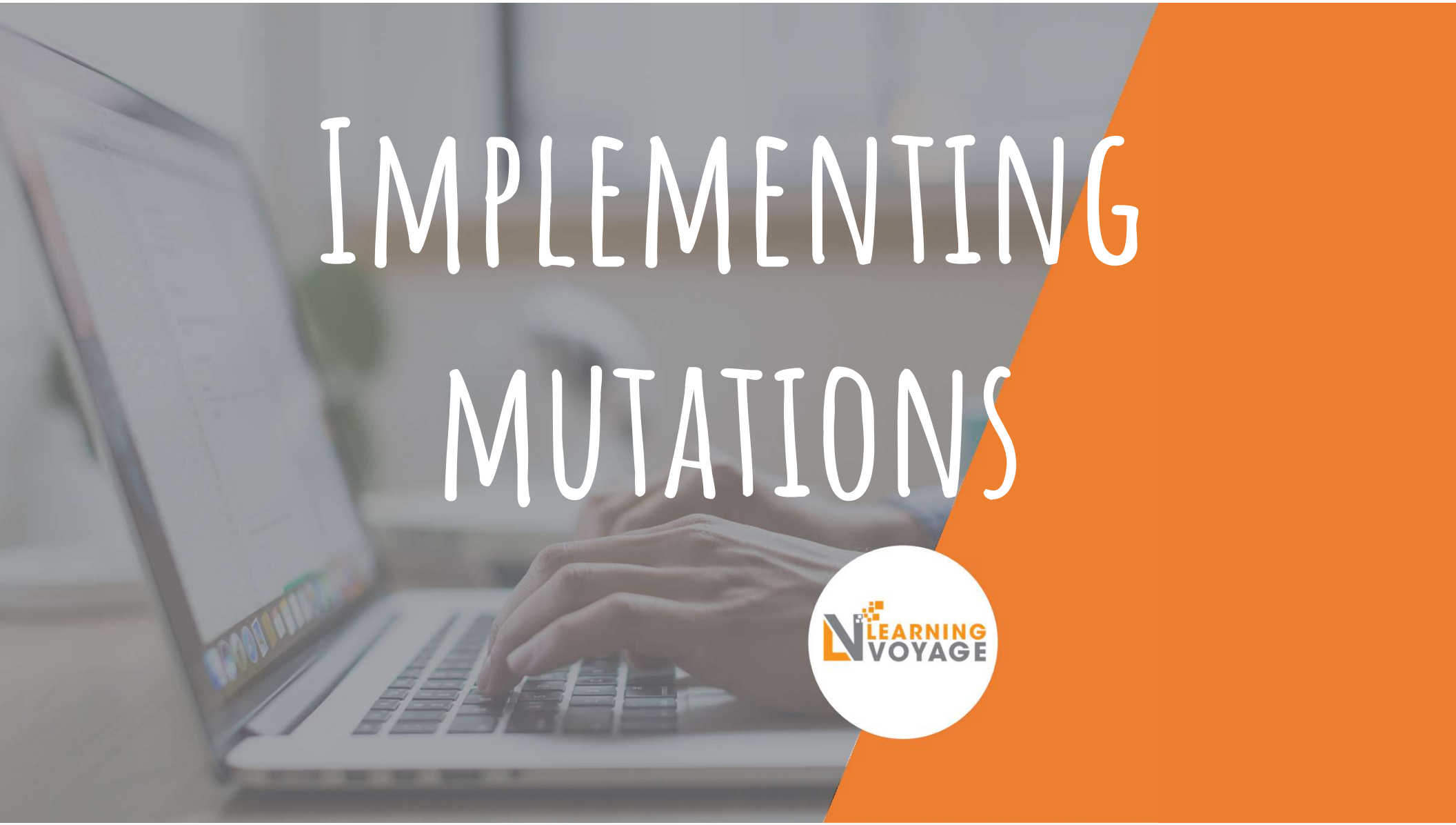
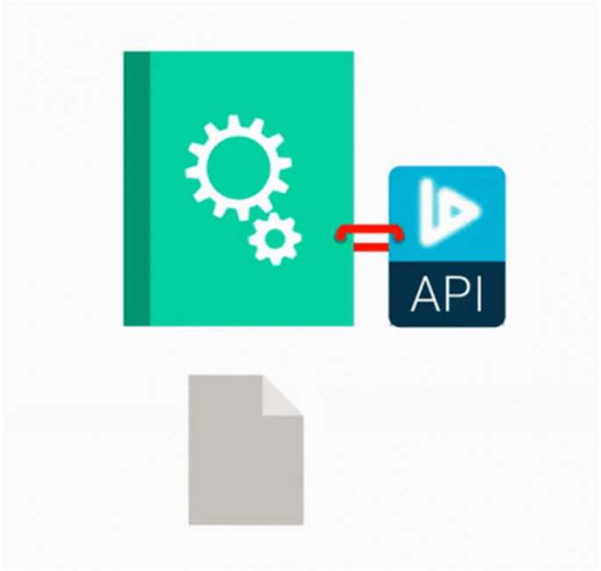

Implementing
mutations
LEARNING
VOYAGE

# Table of Contents

# The mutators context object

```
const pgApiWrapper = async () => {
  // ....

  return {
    // ....

    mutators: {

    },
  };
};
```



# The mutators context object

```
const mongoApiWrapper = async () => {
  // ·-·-·

  return {
    // ·-·-·

    mutators: {

    },
  };
};
```

```
async function main() {
  // ·-·-·

  server.use('/', (req, res) => {
    // ·-·-·

    const mutators = {
      ...pgApi.mutators,
      ...mongoApi.mutators,
    };

    graphqlHTTP({
      schema,
      context: { loaders, mutators },
      graphiql: true,
    })(req, res);
  });

  // ·-·-·
};
```

# The mutators context object

# Table of Contents

# The Mutation type

```
import QueryType from './queries';
import MutationType from './mutations';

export const schema = new GraphQLSchema({
  query: QueryType,
  mutation: MutationType,
});
```

# The Mutation type

```
import { GraphQLObjectType } from 'graphql';
const MutationType = new GraphQLObjectType({
  name: 'Mutation',
  fields: () => ({
    // ·-·-·
  }),
});

export default MutationType;
```

# Table of Contents

# User mutations

```
input UserInput {
  username: String!
  password: String!
  firstName: String
  lastName: String
}

type UserError {
  message: String!
}

type UserPayload {
  errors: [UserError!]!
  user: User
  authToken: String!
}

type Mutation {
  userCreate(input: UserInput!): UserPayload!

  # More mutations
}
```

# User mutations

```
import {
  GraphQLObjectType,
  GraphQLString,
  GraphQLNonNull,
} from 'graphql';

const UserError = new GraphQLObjectType({
  name: 'UserError',
  fields: () => ({
    message: {
      type: new GraphQLNonNull(GraphQLString),
    },
  }),
});

export default UserError;
```

# User mutations

```
import {
  GraphQLObjectType,
  GraphQLString,
  GraphQLNonNull,
  GraphQLList,
} from 'graphql';

import User from './user';
import UserError from './user-error';

const UserPayload = new GraphQLObjectType({
  name: 'UserPayload',
  fields: () => ({
    errors: {
      type: new GraphQLNonNull(
        new GraphQLList(new GraphQLNonNull(UserError)),
      ),
    },
    user: { type: User },
    authToken: { type: GraphQLString },
  }),
});

export default UserPayload;
```

# User mutations

```
import {
  GraphQLInputObjectType,
  GraphQLString,
  GraphQLNonNull,
} from 'graphql';

const UserInput = new GraphQLInputObjectType({
  name: 'UserInput',
  fields: () => ({
    username: { type: new GraphQLNonNull(GraphQLString) },
    password: { type: new GraphQLNonNull(GraphQLString) },
    firstName: { type: GraphQLString },
    lastName: { type: GraphQLString },
  }),
});

export default UserInput;
```

```
import { GraphQLObjectType, GraphQLNonNull } from 'graphql';
import UserPayload from './types/payload-user';
import UserInput from './types/input-user';

const MutationType = new GraphQLObjectType({
  name: 'Mutation',
  fields: () => ({
    userCreate: {
      type: new GraphQLNonNull(UserPayload),
      args: {
        input: { type: new GraphQLNonNull(UserInput) },
      },
      resolve: async (source, { input }, { mutators }) => {
        return mutators.userCreate({ input });
      },
    },
  }),
});

export default MutationType;
```

# User mutations

# User mutations

- We can actually tell PostgreSQL to return a newly created record using the same INSERT statement.

- Here's an example.

```
INSERT INTO azdev.users (username, password)
  VALUES ('janedoe', 'ChangeMe')
RETURNING id, username, created_at
```

```
import { randomString } from '../utils';

const pgApiWrapper = async () => {
  // ·-·-·

  return {
    // ·-·-·

    mutators: {
      // ·-·-·

      userCreate: async ({ input }) => {
        const payload = { errors: [] };
        if (input.password.length < 6) {
          payload.errors.push({
            message: 'Use a stronger password',
          });
        }
        if (payload.errors.length === 0) {
          const authToken = randomString();
          const pgResp = await pgQuery(sqls.userInsert, {
            $1: input.username.toLowerCase(),
            $2: input.password,
            $3: input.firstName,
            $4: input.lastName,
            $5: authToken,
          });
          if (pgResp.rows[0]) {
            payload.user = pgResp.rows[0];
            payload.authToken = authToken;
          }
        }
        return payload;
      },
    }
```

# User mutations

# User mutations

- To test the userCreate mutation, here's a request you can use in GraphiQL.

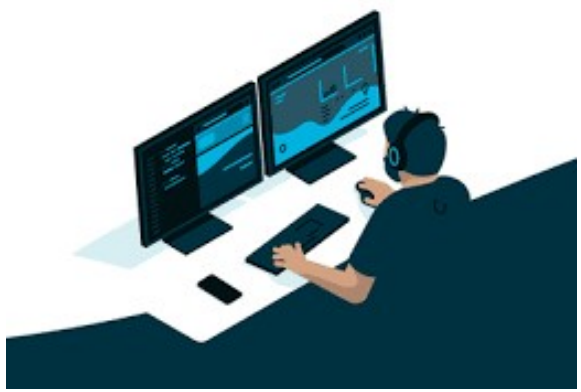
```
mutation userCreate {
  userCreate(input: {
    username: "janedoe"
    password: "123"
    firstName: "Jane"
    lastName: "Doe"
  }) {
    errors {
      message
    }
    user {
      id
      name
    }
    authToken
  }
}
```

# User mutations

```
mutation userCreate {
  userCreate(input: {
    username: "janedoe"
    password: "123"
    firstName: "Jane"
    lastName: "Doe"
  }) {
    errors {
      message
    }
    user {
      id
      name
    }
    authToken
  }
}
```

```
{
  "data": {
    "userCreate": {
      "errors": [
        {
          "message": "Use a stronger password"
        }
      ],
      "user": null,
      "authToken": null
    }
  }
}
```

# The userLogin mutation



```
input AuthInput {
  username: String!
  password: String!
}

type Mutation {
  userLogin(input: AuthInput!): UserPayload!

  # .-.-.
}
```

# The userLogin mutation

```
import {
  GraphQLInputObjectType,
  GraphQLString,
  GraphQLNonNull,
} from 'graphql';

const AuthInput = new GraphQLInputObjectType({
  name: 'AuthInput',
  fields: () => ({
    username: { type: new GraphQLNonNull(GraphQLString) },
    password: { type: new GraphQLNonNull(GraphQLString) },
  }),
});

export default AuthInput;
```

```
// ----

import AuthInput from './types/input-auth';

const MutationType = new GraphQLObjectType({
  name: 'Mutation',
  fields: () => ({
    // ----
    userLogin: {
      type: new GraphQLNonNull(UserPayload),
      args: {
        input: { type: new GraphQLNonNull(AuthInput) },
      },
      resolve: async (source, { input }, { mutators }) => {
        return mutators.userLogin({ input });
      },
    },
  }),
});

export default MutationType;
```

# The userLogin mutation

# The userLogin mutation

```
const pgApiWrapper = async () => {
  // ·-·-·

  return {
    // ·-·-·

    mutators: {
      // ·-·-·
       userLogin: async ({ input }) => {
        const payload = { errors: [] };
        if (!input.username || !input.password) {
          payload.errors.push({
            message: 'Invalid username or password',
          });
        }
        if (payload.errors.length === 0) {
          const pgResp = await pgQuery(sqls.userFromCredentials, {
            $1: input.username.toLowerCase(),
            $2: input.password,
          });
          const user = pgResp.rows[0];
          if (user) {
            const authToken = randomString();
            await pgQuery(sqls.userUpdateAuthToken, {
              $1: user.id,
              $2: authToken,
            });
            payload.user = user;
            payload.authToken = authToken;
          } else {
            payload.errors.push({
              message: 'Invalid username or password'
            });
          }
        }
        return payload;
```

# The userLogin mutation

- To test the userLogin mutation, here's a request you can use in GraphiQL.

```
mutation userLogin {
  userLogin(input: {
    username: "test",
    password: "123456"
  }) {
    errors {
      message
    }
    user {
      id
      name
    }
    authToken
  }
}
```

# The userLogin mutation

```
mutation userLogin {
  userLogin(input: {
    username: "test",
    password: "42"
  }) {
    errors {
      message
    }
    user {
      id
      name
    }
    authToken
  }
}
```

```
{
  "data": {
    "userLogin": {
      "errors": [
        {
          "message": "Invalid username or password"
        }
      ],
      "user": null,
      "authToken": null
    }
  }
}
```

# The userLogin mutation

```
mutation userLogin {
  userLogin(input: {
    username: "test",
    password: "123456"
  }) {
    errors {
      message
    }
    user {
      id
      name
    }
    authToken
  }
}
```

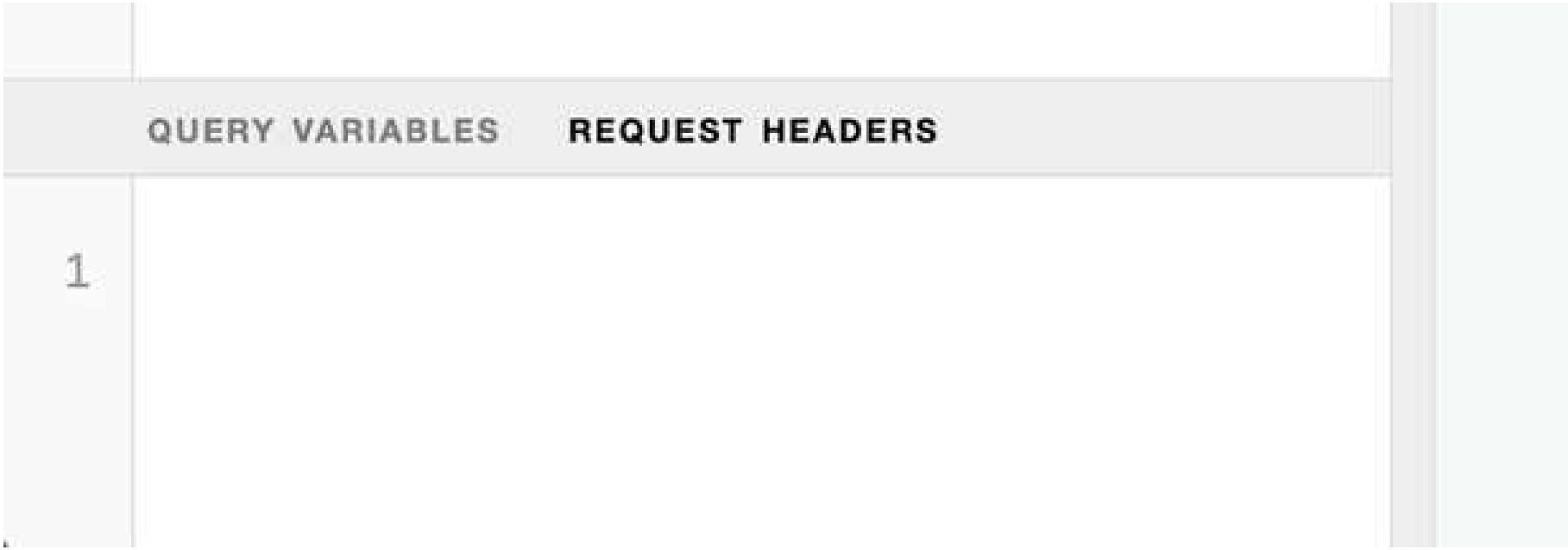
```
{
  "data": {
    "userLogin": {
      "errors": [],
      "user": {
        "id": "1",
        "name": ""
      },
      "authToken": "078a4a415c12a88af7bd35f6ec8be
    }
  }
}
```

# Table of Contents

```
async function main() {
  // ·-·-·

  server.use('/', (req, res) => {
    // ·-·-·
    graphqlHTTP({
      schema,
      context: { loaders, mutators },
      graphiql: { headerEditorEnabled: true },
      // ·-·-·
    })(req, res);
  });

  // ·-·-·
}
```

# Authenticating API consumers

# Authenticating API consumers



# Authenticating API consumers

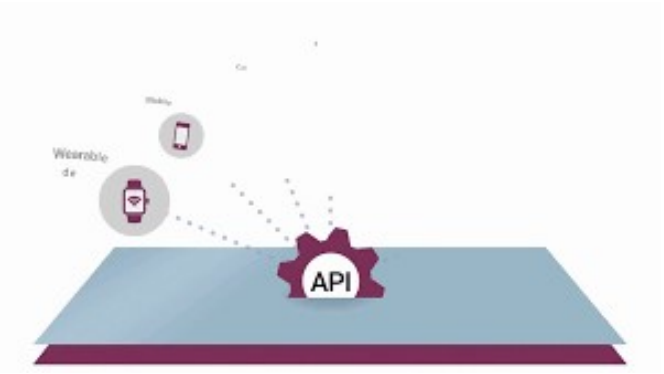
- We can use the Authorization request header to include the authToken value with every request made by GraphiQL.
- The syntax for that request header is shown next.

```
Authorization: <type> <credentials>
```

# Authenticating API consumers

- Use the userLogin mutation to obtain a valid authToken value using the “test/123456” credentials.
- That user owns a private Task record that should show up if they send the following GraphQL query.

```
{
  search(term: "babel") {
    content
  }
}
```



# Authenticating API consumers

- To include the authToken value, put this in the request headers editor:

```
{
  "Authorization": "Bearer AUTH_TOKEN_VALUE_HERE"
}
```

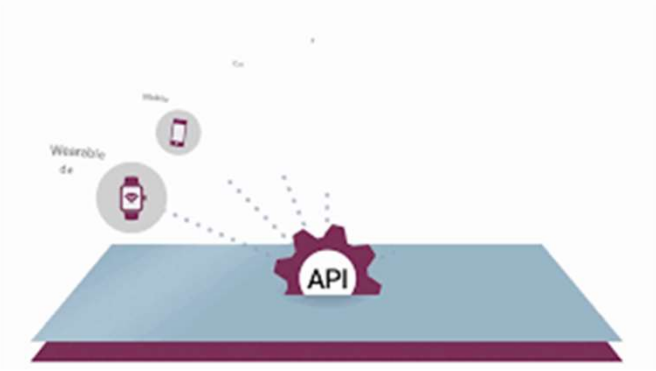
```
const pgApiWrapper = async () => {
  // ·-·-·

  return {
    userFromAuthToken: async (authToken) => {
      if (!authToken) {
        return null;
      }
      const pgResp = await pgQuery(sqls.userFromAuthToken, {
        $1: authToken,
      });
      return pgResp.rows[0];
    },

    // ·-·-·
  };
};
```

# Authenticating API consumers

```
async function main() {
  // ...

  server.use('/', async (req, res) => {
    const authToken =
      req && req.headers && req.headers.authorization
        ? req.headers.authorization.slice(7) // "Bearer "
        : null;
    const currentUser = await pgApi.userFromAuthToken(authToken);
    if (authToken && !currentUser) {
      return res.status(401).send({
        errors: [{ message: 'Invalid access token' }],
      });
    }

    // ...
  });

  // ...
}
```

# Authenticating API consumers

```
async function main() {
  // ·-·-·

  server.use('/', async (req, res) => {
    // ·-·-·

    const loaders = {
      users: new DataLoader((userIds) => pgApi.usersInfo(userIds)),
      approachLists: new DataLoader((taskIds) =>
        pgApi.approachLists(taskIds),
      ),
      tasks: new DataLoader((taskIds) =>
        pgApi.tasksInfo({ taskIds, currentUser }),
      ),
      tasksByTypes: new DataLoader((types) =>
        pgApi.tasksByTypes(types),
      ),
      searchResults: new DataLoader((searchTerms) =>
        pgApi.searchResults({ searchTerms, currentUser }),
      ),
      detailLists: new DataLoader((approachIds) =>
        mongoApi.detailLists(approachIds),
      ),
    };

    // ·-·-·
  });

  // ·-·-·
}
```

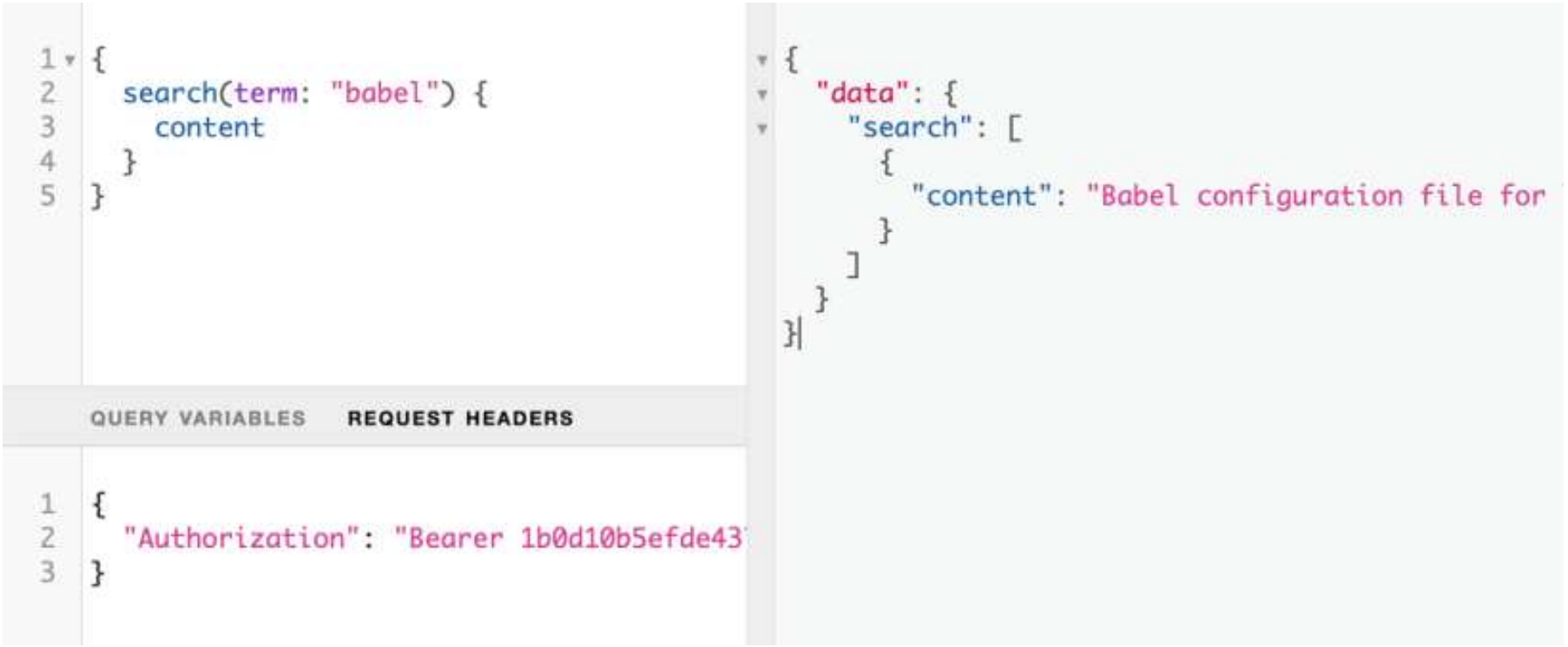



# Authenticating API consumers

# Authenticating API consumers

```
const pgApiWrapper = async () => {
  // ----

  return {
    // ----
    tasksInfo: async ({ taskIds, currentUser }) => {
      const pgResp = await pgQuery(sqls.tasksFromIds, {
        $1: taskIds,
        $2: currentUser ? currentUser.id : null,
      });
      return taskIds.map((taskId) =>
        pgResp.rows.find((row) => taskId == row.id),
      );
    },
    searchResults: async ({ searchTerms, currentUser }) => {
      const results = searchTerms.map(async (searchTerm) => {
        const pgResp = await pgQuery(sqls.searchResults, {
          $1: searchTerm,
          $2: currentUser ? currentUser.id : null,
        });
        return pgResp.rows;
      });
      return Promise.all(results);
    },

    // ----
  };
};
```

# Sending a request header with a GraphQL query



# Testing an invalid Authorization request header

```
{
  search(term: "babel") {
    content
  }
}
```

QUERY VARIABLES **REQUEST HEADERS**

```
{
  "Authorization": "Bearer FAKE_TOKEN"
}
```

```
{
  "errors": [
    {
      "message": "Invalid access token"
    }
  ]
}
```

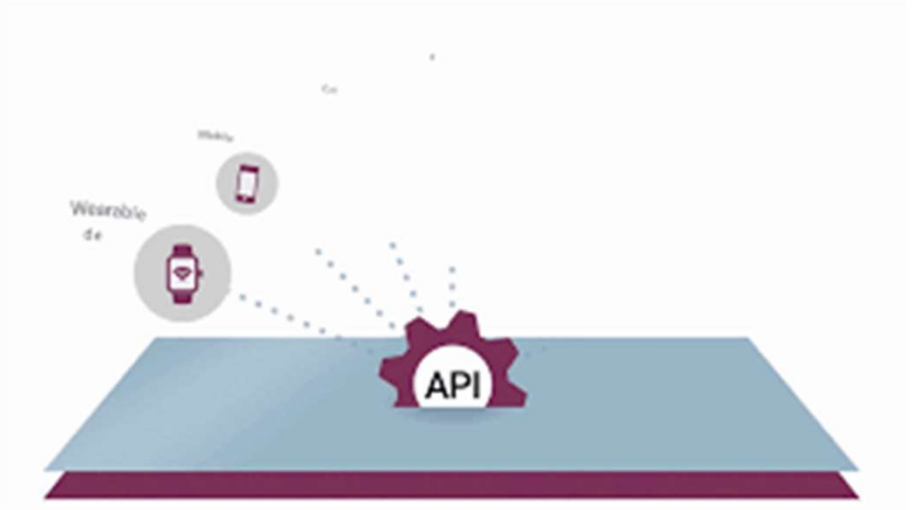
```
type Query {
  // ·-·-·
  me: User
}

type User {
  id: ID!
  createdAt: String!
  username: String!
  name: String
  taskList: [Task!]!
}
```

# The me root query field

# The me root query field

- Let's start with the me field itself. Here's a query we can use to test it when it's finished.

```
{
  me {
    id
    username
  }
}
```



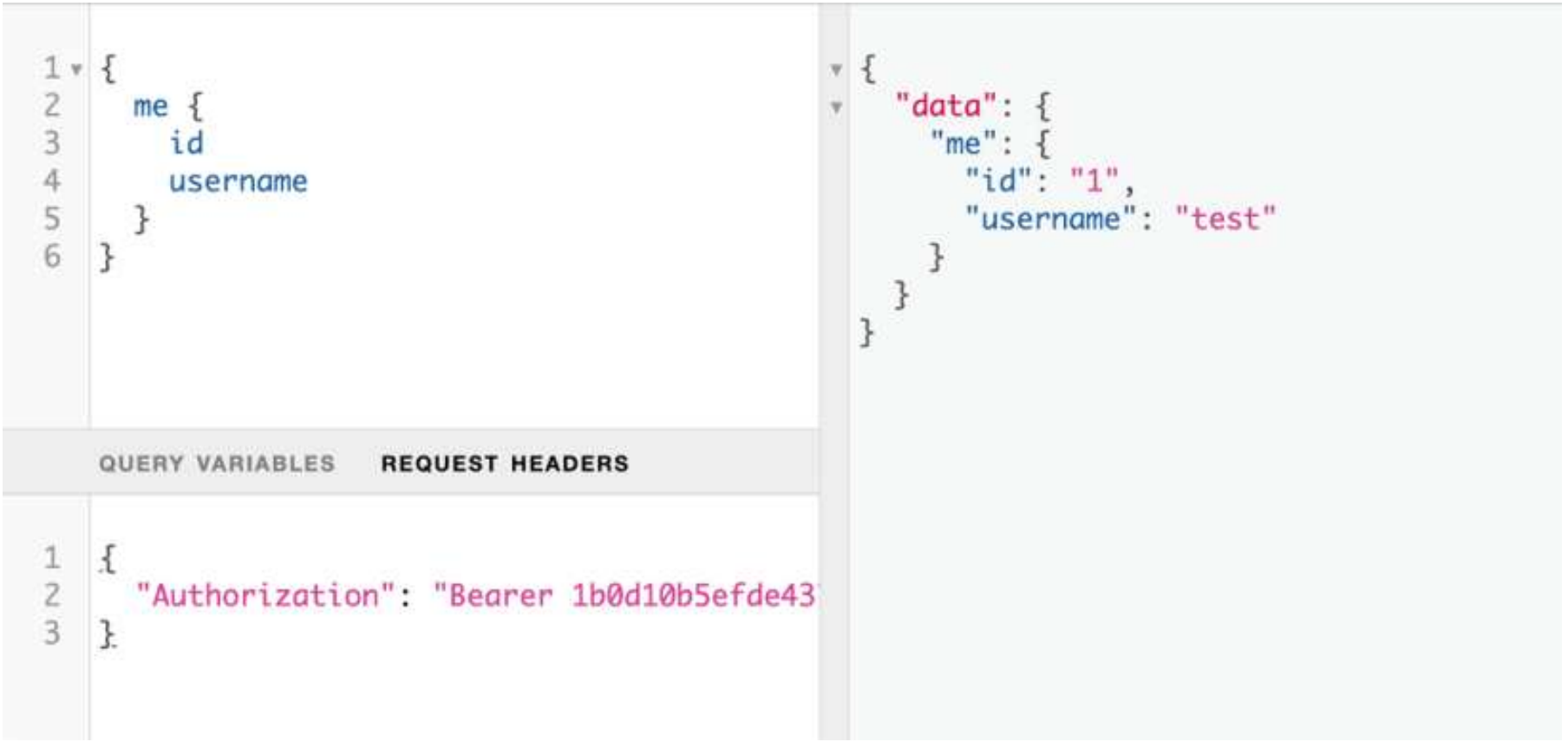
# The me root query field

```
async function main() {
  // ·-·-·

  server.use('/', async (req, res) => {
    // ·-·-·
    graphqlHTTP({
      schema,
      context: { loaders, mutators, currentUser },
      graphiql: { headerEditorEnabled: true },
      // ·-·-·
    })(req, res);
  });

  // ·-·-·
}
```

# The me root query field

```
// ·-·-·
import User from './types/user';

const QueryType = new GraphQLObjectType({
  name: 'Query',
  fields: () => ({
    // ·-·-·
    me: {
      type: User,
      resolve: async (source, args, { currentUser }) => {
        return currentUser;
      },
    },
  }),
});
```

# The me root query field



# The me root query field

```
import {
  // ...
  GraphQLList,
} from 'graphql';

import Task from './task';

const fieldsWrapper = ({ meScope }) => {
  const userFields = {
    id: { type: new GraphQLNonNull(GraphQLID) },
    username: { type: GraphQLString },
    name: {
      type: GraphQLString,
      resolve: ({ firstName, lastName }) =>
        [firstName, lastName].filter(Boolean).join(' '),
    },
  };
```

# Continued code

```
  if (meScope) {
    userFields.taskList = {
      type: new GraphQLNonNull(
        new GraphQLList(new GraphQLNonNull(Task)),
      ),
      resolve: (source, args, { loaders, currentUser }) => {
        return loaders.tasksForUsers.load(currentUser.id);
      },
    };
  }

  return userFields;
};
const User = new GraphQLObjectType({
  name: 'User',
  fields: () => fieldsWrapper({ meScope: false }),
});

export const Me = new GraphQLObjectType({
  name: 'Me',
  fields: () => fieldsWrapper({ meScope: true }),
});

export default User;
```

# HOW TO USE THE NEW ME TYPE.

```
// ....
import { Me } from './types/user';

const QueryType = new GraphQLObjectType({
  name: 'Query',
  fields: () => ({
    // ....
     me: {
      type: Me,
      resolve: async (source, args, { currentUser }) => {
        return currentUser;
      },
    },
  }),
});
```

# The me root query field

- Let's define the loaders.tasksForUsers function and its batch-loading pgApi function.

```
const loaders = {
  // ·-·-·

  tasksForUsers: new DataLoader((userIds) =>
    pgApi.tasksForUsers(userIds),
  ),
};
```

# The me root query field

```
const pgApiWrapper = async () => {
  // ....

  return {
    // ....

    tasksForUsers: async (userIds) => {
      const pgResp = await pgQuery(sqls.tasksForUsers, {
        $1: userIds,
      });
      return userIds.map((userId) =>
        pgResp.rows.filter((row) => userId === row.userId),
      );
    },

    // ....
  };
};
```

# The me root query field

```
{
  me {
    id
    username
    taskList {
      content
    }
  }
}
```

QUERY VARIABLES  REQUEST HEADERS

```
{
  "Authorization": "Bearer 1b0d10b5efde43
}
```


```
{
  "data": {
    "me": {
      "id": "1",
      "username": "test",
      "taskList": [
        {
          "content": "Make an image in HTML change based on the theme color mode (dark or light)"
        },
        {
          "content": "Get rid of only the unstaged changes since the last git commit"
        },
        {
          "content": "The syntax for a switch statement (AKA case statement) in JavaScript"
        },
        {
          "content": "Calculate the sum of numbers in a JavaScript array"
        },
        {
```

# The me root query field

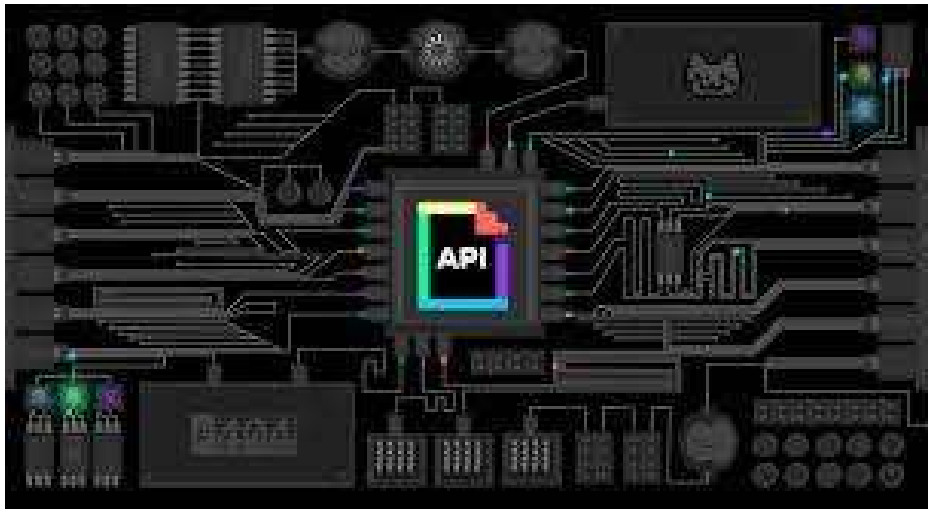
```
{
  me {
    id
    username
    taskList {
      content
    }
  }
}
```

```
{
  taskMainList {
    content
    author {
      username
      taskList {
        content
      }
    }
  }
}
```

```
{
  "errors": [
    {
      "message": "Cannot query field \"taskList\" on type \"User\".",
      "locations": [
        {
          "line": 6,
          "column": 7
        }
      ],
      "stack": [
        "GraphQLError: Cannot query field \"taskList\" on type \"User\".",
```

# The me root query field

```
{
  taskMainList {
    content
    author {
      username
      taskList {
        content
      }
    }
  }
}
```



# Table of Contents

```
input TaskInput {
  content: String!
  tags: [String!]!
  isPrivate: Boolean!
}

type TaskPayload {
  errors: [UserError!]!
  task: Task
}

type Mutation {
  taskCreate(input: TaskInput!): TaskPayload!

  # ·-·-·
}
```

# Mutations for the Task model

# Mutations for the Task model

```
import {
  GraphQLInputObjectType,
  GraphQLString,
  GraphQLNonNull,
  GraphQLBoolean,
  GraphQLList,
} from 'graphql';

const TaskInput = new GraphQLInputObjectType({
  name: 'TaskInput',
  fields: () => ({
    content: { type: new GraphQLNonNull(GraphQLString) },
    tags: {
      type: new GraphQLNonNull(
        new GraphQLList(new GraphQLNonNull(GraphQLString)),
      ),
    },
    isPrivate: { type: new GraphQLNonNull(GraphQLBoolean) },
  }),
});

export default TaskInput;
```

# Mutations for the Task model

```
import {
  GraphQLObjectType,
  GraphQLNonNull,
  GraphQLList,
} from 'graphql';

import Task from './task';
import UserError from './user-error';

const TaskPayload = new GraphQLObjectType({
  name: 'TaskPayload',
  fields: () => ({
    errors: {
      type: new GraphQLNonNull(
        new GraphQLList(new GraphQLNonNull(UserError)),
      ),
    },
    task: { type: Task },
  }),
});

export default TaskPayload;
```

# Mutations for the Task model

```
// ...-.-.
import TaskPayload from './types/payload-task';
import TaskInput from './types/input-task';

const MutationType = new GraphQLObjectType({
  name: 'Mutation',
  fields: () => ({
    // ...-.-.

    taskCreate: {
      type: TaskPayload,
      args: {
        input: { type: new GraphQLNonNull(TaskInput) },
      },
      resolve: async (
        source,
        { input },
        { mutators, currentUser },
      ) => {
        return mutators.taskCreate({ input, currentUser });
      },
    },
  }),
});
```

# Mutations for the Task model

```
const pgApiWrapper = async () => {
  // ...

  return {
    // ...
    mutators: {
      // ...
      taskCreate: async ({ input, currentUser }) => {
        const payload = { errors: [] };
        if (input.content.length < 15) {
          payload.errors.push({
            message: 'Text is too short',
          });
        }
        if (payload.errors.length === 0) {
          const pgResp = await pgQuery(sqls.taskInsert, {
            $1: currentUser.id,
            $2: input.content,
            $3: input.tags.join(','),
            $4: input.isPrivate,
          });

          if (pgResp.rows[0]) {
            payload.task = pgResp.rows[0];
          }
        }

        return payload;
      },
    },
  };
};
```

```
mutation taskCreate {
  taskCreate (
    input: {
      content: "Use INSERT/SELECT together in PostgreSQL",
      tags: ["sql", "postgresql"]
      isPrivate: false,
   }
  ) {
    errors {
      message
    }
    task {
      id
      content
      tags
      author {
        id
      }
      createdAt
    }
  }
}
```

# Mutations for the Task model

# Table of Contents

# The approachCreate mutation

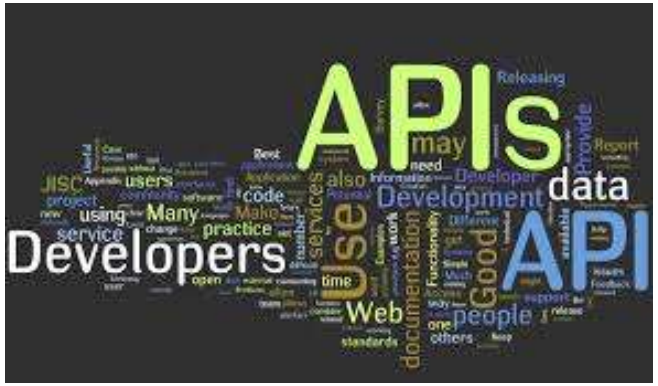
```
input ApproachDetailInput {
  content: String!
  category: ApproachDetailCategory!
}

input ApproachInput {
  content: String!
  detailList: [ApproachDetailInput!]!
}

type ApproachPayload {
  errors: [UserError!]!
  approach: Approach
}

type Mutation {
  approachCreate(
    taskId: ID!
    input: ApproachInput!
  ): ApproachPayload!

  # ·-·-·
}
```

```
import {
  // ·-·-·
  GraphQLID,
} from 'graphql';
// ·-·-·
import ApproachPayload from './types/payload-approach';
import ApproachInput from './types/input-approach';

const MutationType = new GraphQLObjectType({
  name: 'Mutation',
  fields: () => ({
    // ·-·-·

    approachCreate: {
      type: ApproachPayload,
      args: {
        taskId: { type: new GraphQLNonNull(GraphQLID) },
        input: { type: new GraphQLNonNull(ApproachInput) },
      },
      resolve: async (
        source,
        { taskId, input },
        { mutators, currentUser },
      ) => {
        return mutators.approachCreate({
          taskId,
          input,
          currentUser,
          mutators,
        });
      },
    },
  }),
});
```

# The approachCreate mutation

# The approachCreate mutation

```
import {
  GraphQLList,
  GraphQLNonNull,
  GraphQLObjectType,
} from 'graphql';

import Approach from './approach';
import UserError from './user-error';

const ApproachPayload = new GraphQLObjectType({
  name: 'ApproachPayload',
  fields: () => ({
    errors: {
      type: new GraphQLNonNull(
        new GraphQLList(new GraphQLNonNull(UserError)),
      ),
    },
    approach: { type: Approach },
  }),
});

export default ApproachPayload;
```



# The approachCreate mutation

```
import {
  GraphQLInputObjectType,
  GraphQLString,
  GraphQLNonNull,
} from 'graphql';

import ApproachDetailCategory from './approach-detail-category';

const ApproachDetailInput = new GraphQLInputObjectType({
  name: 'ApproachDetailInput',
  fields: () => ({
    content: { type: new GraphQLNonNull(GraphQLString) },
    category: {
      type: new GraphQLNonNull(ApproachDetailCategory),
    },
  }),
});

export default ApproachDetailInput;
```

# The approachCreate mutation

```
import {
  GraphQLInputObjectType,
  GraphQLString,
  GraphQLNonNull,
  GraphQLList,
} from 'graphql';

import ApproachDetailInput from './input-approach-detail';

const ApproachInput = new GraphQLInputObjectType({
  name: 'ApproachInput',
  fields: () => ({
    content: { type: new GraphQLNonNull(GraphQLString) },
    detailList: {
      type: new GraphQLNonNull(
        new GraphQLList(new GraphQLNonNull(ApproachDetailInput)),
      ),
    },
  }),
});

export default ApproachInput;
```

# The approachCreate mutation

```
const pgApiWrapper = async () => {
  // ·-·-·

  return {
    // ·-·-·

    mutators: {
      // ·-·-·

      approachCreate: async ({
        taskId,
        input,
        currentUser,
        mutators,
      }) => {
        const payload = { errors: [] };
          if (payload.errors.length === 0) {
            const pgResp = await pgQuery(sqls.approachInsert, {
            $1: currentUser.id,
            $2: input.content,
            $3: taskId,
          });
          if (pgResp.rows[0]) {
            payload.approach = pgResp.rows[0];
            await pgQuery(sqls.approachCountIncrement, {
              $1: taskId,
            });
            await mutators.approachDetailCreate(
              payload.approach.id,
              input.detailList,
            );
          }
        }

        return payload;
```

```
[
  {
    content: explanationsValue1,
    category: "EXPLANATION"
  },
  {
    content: notesValue1,
    category: "NOTE"
  },
  {
    content: warningsValue1,
    category: "WARNING"
  },
  .-.-.
]
```

# The approachCreate mutation

# The approachCreate mutation

- The format we will convert it into, which is expected by the approachDetails MongoDB collection:

```
{
  explanations: [explanationsValue1, ·-·-·],
  notes: [notesValue1, ·-·-·],
  warnings: [warningsValue1, ·-·-·],
}
```

```
const mongoApiWrapper = async () => {
  // ·-·-·

  return {
    // ·-·-·

    mutators: {
      approachDetailCreate: async (approachId, detailsInput) => {
        const details = {};
        detailsInput.forEach(({ content, category }) => {
          details[category] = details[category] || [];
          details[category].push(content);
        });
        return mdb.collection('approachDetails').insertOne({
          pgId: approachId,
          ...details,
        });
      },
    },
  };
};
```

# The approachCreate mutation

# The approachCreate mutation

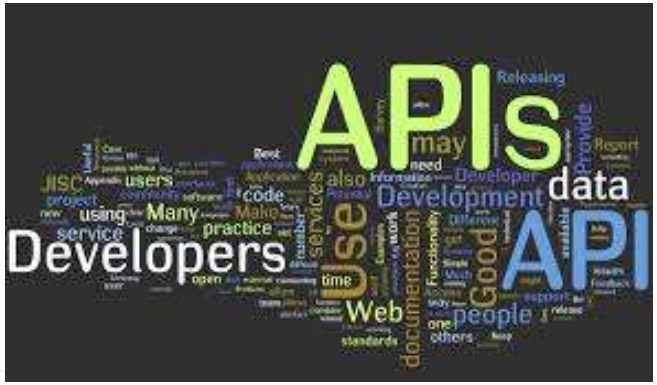
```
const ApproachDetailCategory = new
GraphQLEnumType({
  name: 'ApproachDetailCategory',
  values: {
    NOTE: { value: 'notes' },
    EXPLANATION: { value: 'explanations' },
    WARNING: { value: 'warnings' },
  },
});
```

```
const mongoApiWrapper = async () => {
  // ·-·-·

  return {
    detailLists: async (approachIds) => {
      // ·-·-·
      return approachIds.map((approachId) => {
        // ·-·-·
        if (explanations) {
          approachDetails.push(
            ...explanations.map((explanationText) => ({
              content: explanationText,
              category: 'explanations',
            }))
          );
        }
        if (notes) {
          approachDetails.push(
            ...notes.map((noteText) => ({
              content: noteText,
              category: 'notes',
            }))
          );
        }
        if (warnings) {
          approachDetails.push(
            ...warnings.map((warningText) => ({
              content: warningText,
              category: 'warnings',
            }))
          );
        }
        return approachDetails;
      });
    },
```

# The approachCreate mutation

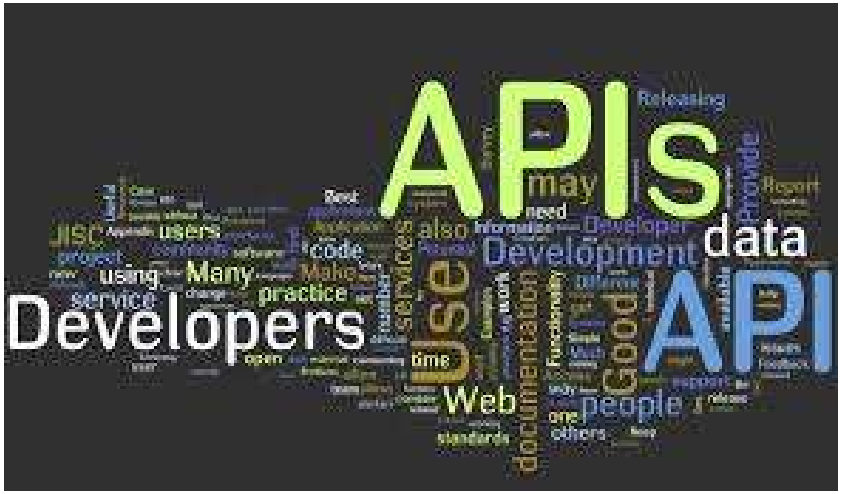
# TEST THE APPROACHCREATE MUTATION

```
mutation approachCreate {
  approachCreate(
    taskId: 42 # Get this value from a taskCreate mutation call
    input: {
      content: "INSERT INTO tableName ·-·-·] ) ] SELECT-STATEMENT",
      detailList: [
        {
          content: "You can still use a RETURNING clause after that",
          category: NOTE,
        },
        {
          content: "The INSERT statement only works if the SELECT statement
          ➥ does",
          category: EXPLANATION,
        },
      ],
    }
  ) {
    errors {
      message
    }
    approach {
      id
      content
      voteCount
      author {
        username
      }
      detailList {
        content
        category
```



# The approachVote mutation

```
input ApproachVoteInput {
  """true for up-vote and false for down-vote"""
  up: Boolean!
}

type Mutation {
  approachVote(
    approachId: ID!
    input: ApproachVoteInput!
  ): ApproachPayload!

  # .-.-.
}
```



# The approachVote mutation

```
// $1: approachId
// $2: voteIncrement
approachVote: `
  UPDATE azdev.approaches
  SET vote_count = vote_count + $2
  WHERE id = $1
  RETURNING id, content, ·-·-·;
`,
```

```
mutation approachVote {
  approachVote(
    approachId: 42 # Get this value from approachCreate
    input: { up: false }
  ) {
    errors {
      message
    }
    approach {
      content
      voteCount
    }
  }
}
```

# The approachVote mutation

# The approachVote mutation

```
import {
  GraphQLInputObjectType,
  GraphQLBoolean,
  GraphQLNonNull,
} from 'graphql';

const ApproachVoteInputType = new GraphQLInputObjectType({
  name: 'ApproachVoteInput',
  description: "true for up-vote and false for down-vote",
  fields: () => ({
    up: { type: new GraphQLNonNull(GraphQLBoolean) },
  }),
});

export default ApproachVoteInputType;
```

# The approachVote mutation

```
// ·-·-·
import ApproachVoteInput from './types/input-approach-vote';

const MutationType = new GraphQLObjectType({
  name: 'Mutation',
  fields: () => ({
    // ·-·-·
    approachVote: {
      type: ApproachPayload,
      args: {
        approachId: { type: new GraphQLNonNull(GraphQLID) },
        input: { type: new GraphQLNonNull(ApproachVoteInput) },
      },
      resolve: async (
        source,
        { approachId, input },
        { mutators },
      ) => {
        return mutators.approachVote({ approachId, input });
      },
    },
  }),
});
```

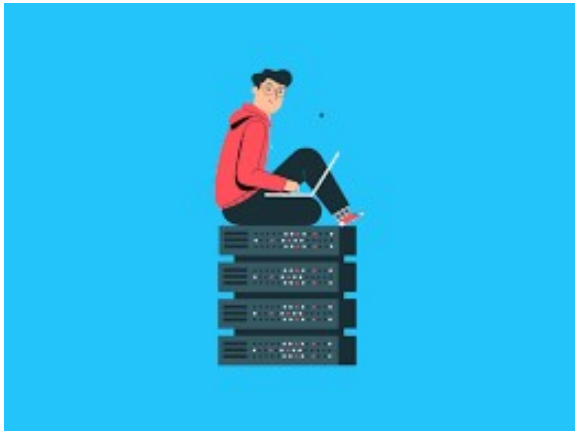
```
const pgApiWrapper = async () => {
  // ·-·-·

  return {
    // ·-·-·
    mutators: {
      // ·-·-·
      approachVote: async ({ approachId, input }) => {
        const payload = { errors: [] };
        const pgResp = await pgQuery(sqls.approachVote, {
          $1: approachId,
          $2: input.up ? 1 : -1,
        });

        if (pgResp.rows[0]) {
          payload.approach = pgResp.rows[0];
        }

        return payload;
      },
    },
  };
};
```

# The approachVote mutation

# Table of Contents

# The userDelete mutation

```
type UserDeletePayload {
  errors: [UserError!]!
  deletedUserId: ID
}

type Mutation {
  userDelete: UserDeletePayload!

  # ·-·-·
}
```



# The userDelete mutation

```
import {
  GraphQLList,
  GraphQLNonNull,
  GraphQLObjectType,
  GraphQLID,
} from 'graphql';

import UserError from './user-error';

const UserDeletePayload = new GraphQLObjectType({
  name: 'UserDeletePayload',
  fields: () => ({
    errors: {
      type: new GraphQLNonNull(
        new GraphQLList(new GraphQLNonNull(UserError)),
      ),
    },
    deletedUserId: { type: GraphQLID },
  }),
});

export default UserDeletePayload;
```

# The userDelete mutation

```
const pgApiWrapper = async () => {
  // ....

  return {
    // ....

    mutators: {
      // ....

      userDelete: async ({ currentUser }) => {
        const payload = { errors: [] };
        try {
          await pgQuery(sqls.userDelete, {
            $1: currentUser.id,
          });
          payload.deletedUserId = currentUser.id;
        } catch (err) {
          payload.errors.push({
            message: 'We were not able to delete this account',
          });
        }

        return payload;
      },
    },
  };
};
```

# The userDelete mutation

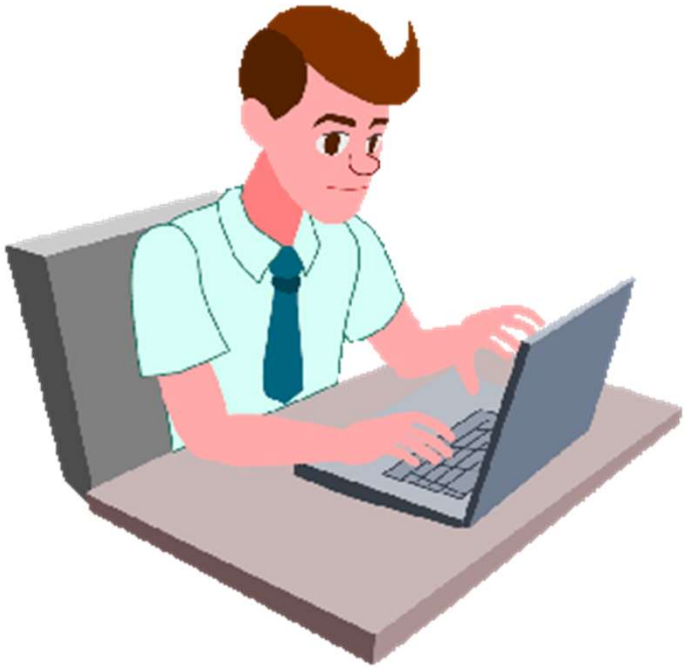
```
// ·-·-·
import UserDeletePayload from './types/payload-user-delete';

const MutationType = new GraphQLObjectType({
  name: 'Mutation',
  fields: () => ({
    // ·-·-·

    userDelete: {
      type: UserDeletePayload,
      resolve: async (source, args, { mutators, currentUser }) => {
        return mutators.userDelete({ currentUser });
      },
    },
  }),
});
```

# The userDelete mutation

```
mutation userDelete {
  userDelete {
    errors {
      message
    }
    deletedUserId
  }
}
```

# Summary

- To host mutations, a GraphQL schema must define a root mutation type.
- To organize database operations for mutations, you can group them on a single object that you expose as part of the global context for resolvers.



# "Complete Lab"