Translator Words Application on Javascript

By @AntonKalik

Where is the source of idea?

worddeposit.com

by @maxkalik

Server / Client

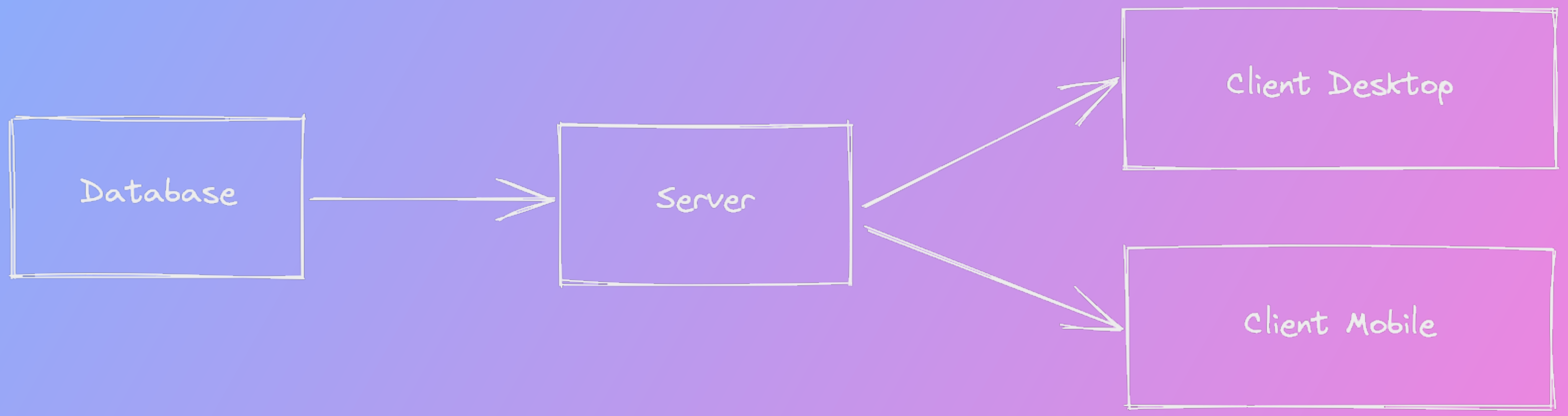
Node / React

Where to start?

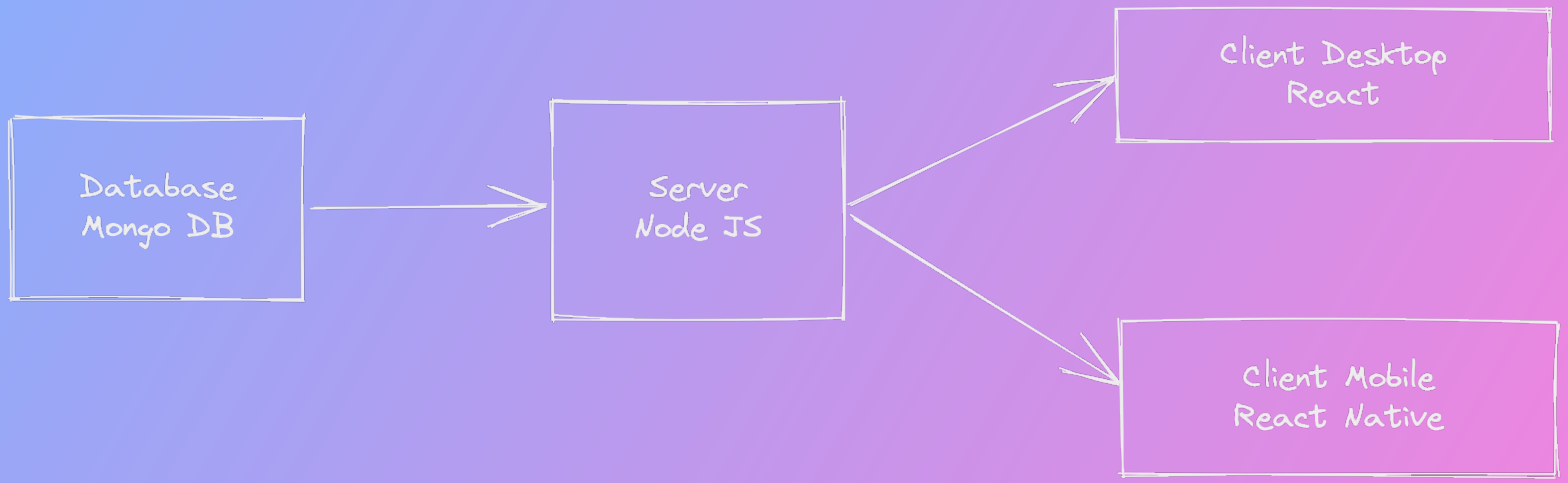
Basic logic / algorithm

Scalable architecture

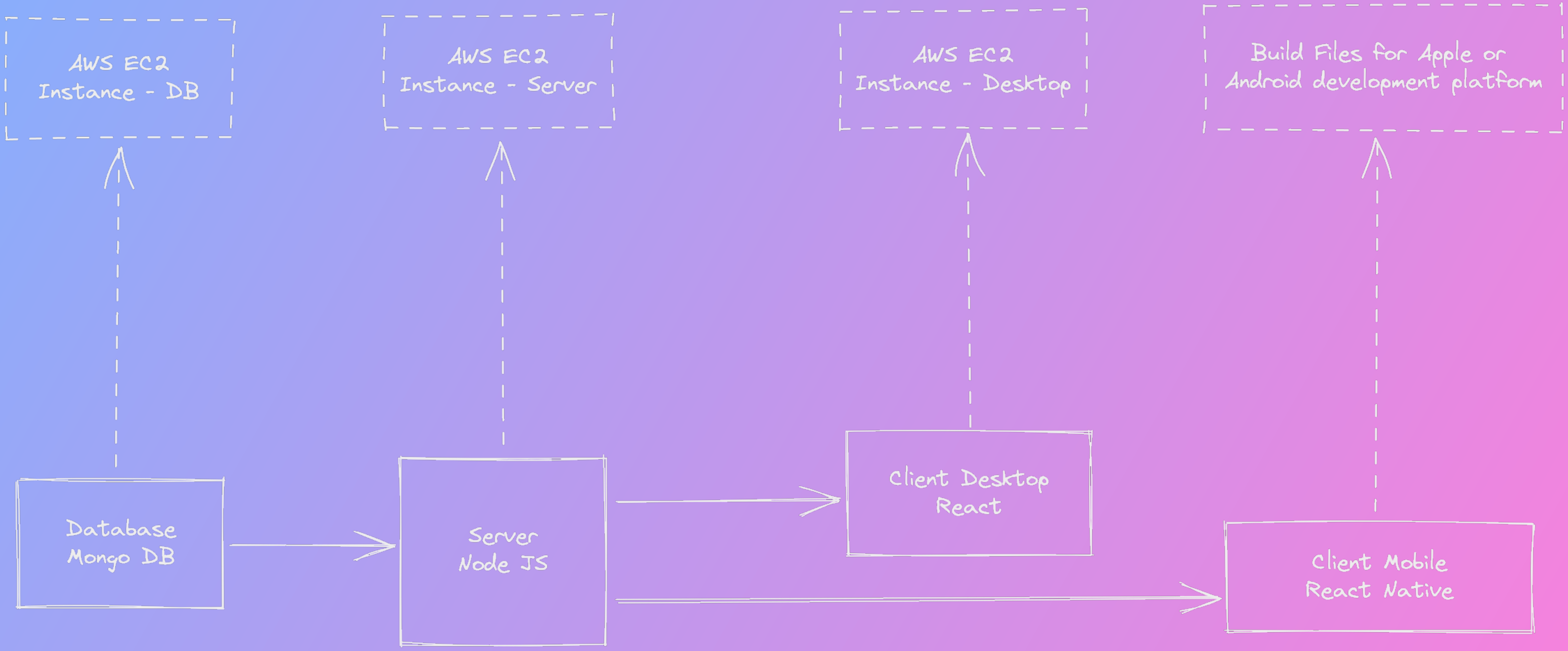
Schema



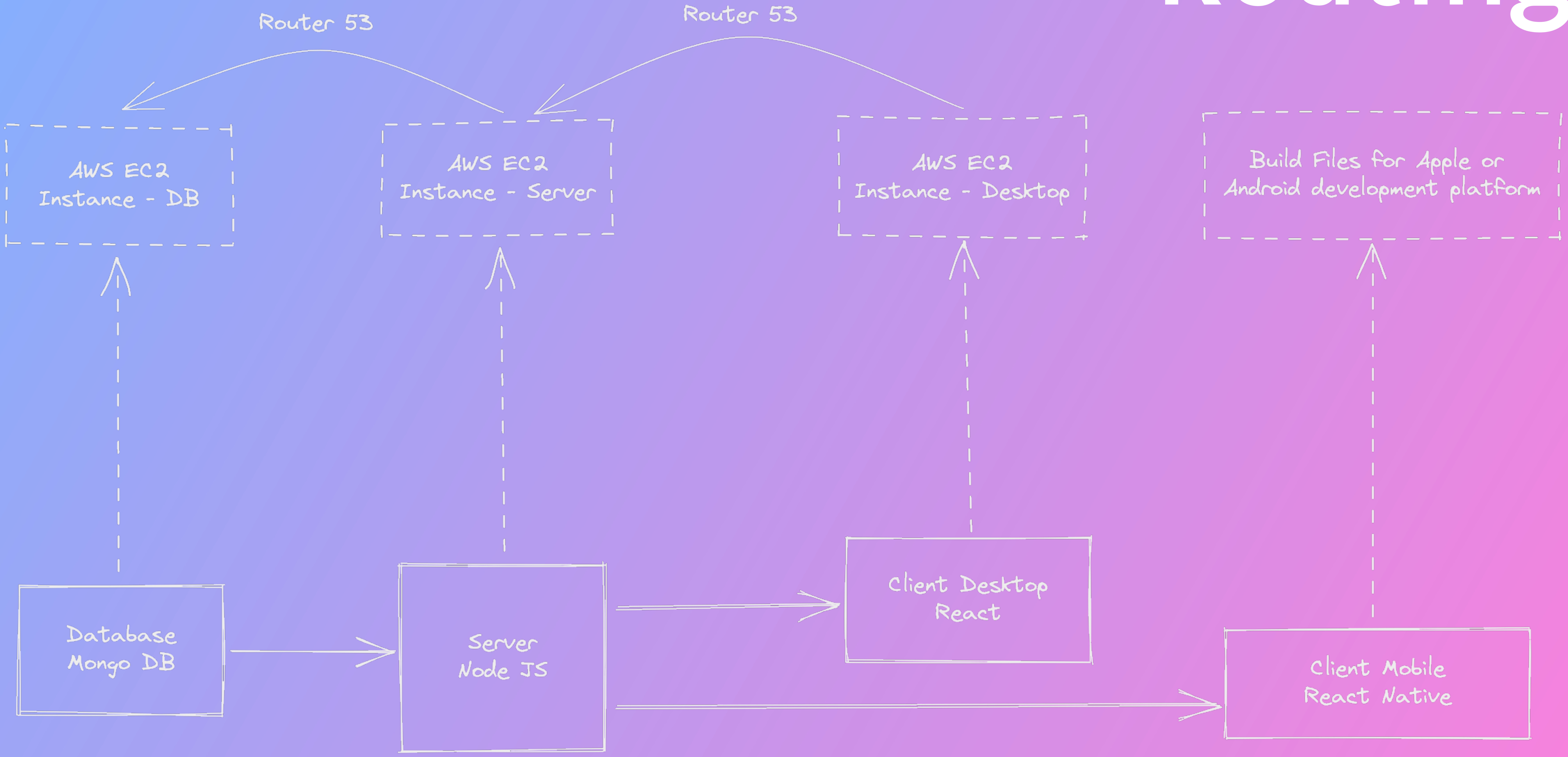
Instruments



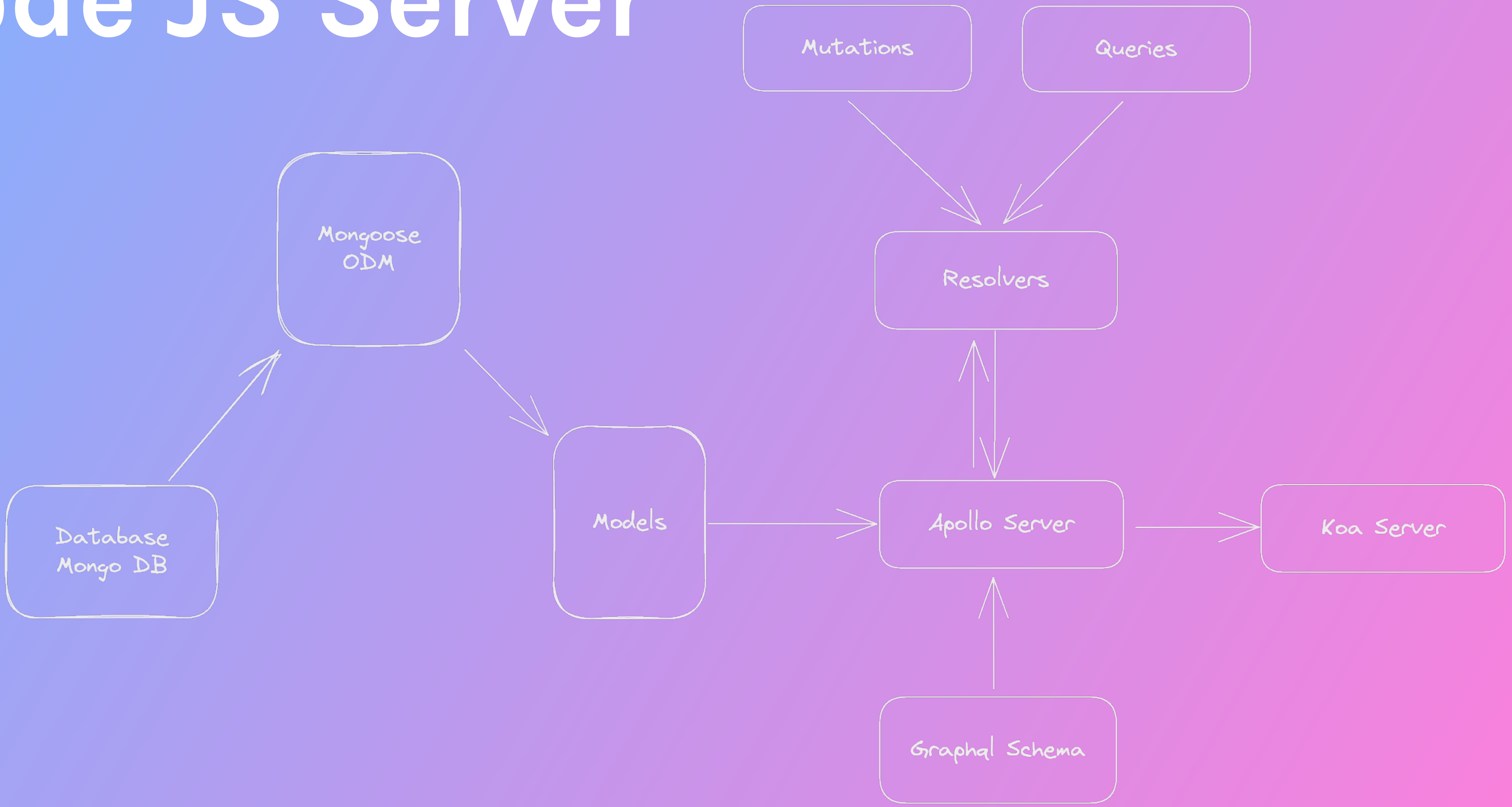
Cloud


AWS EC2
Instance - DB
AWS EC2
Instance - Server
AWS EC2
Instance - Desktop
Build Files for Apple or
Android development platform
Database
Mongo DB
Server
Node JS
Client Desktop
React
Client Mobile
React Native

Routing


Router 53
Router 53
AWS EC2 Instance - DB
AWS EC2 Instance - Server
AWS EC2 Instance - Desktop
Build Files for Apple or Android development platform
Database Mongo DB
Server Node JS
Client Desktop React
Client Mobile React Native

Node JS Server



Server

Docker with Mongo

```
docker-compose up --build mongo
```

Koa Server Launch

```
const app = new Koa();
export const database = new Database();

database.connect().catch(error => {
  console.error('[SERVER] Mongo DB connection Error', error);
  process.exit(1);
});

app.use(cors());
app.use(bodyParser());
app.use(graphqlUploadKoa({ maxFileSize: 10000000, maxFiles: 10 }));

app.on('error', error => {
  console.error('[SERVER] Server Error', error);
});

app.listen(process.env.PORT || 9999);
```

Koa Server Launch

```
apolloServer
  .start()
  .then(() => {
    apolloServer.applyMiddleware({
      app,
      path: '/api/v1/graphql',
    });
  })
  .catch(error => {
    console.error('[SERVER] Apollo Server Error', error);
    process.exit(1);
  });

app.on('error', error => {
  console.error('[SERVER] Server Error', error);
});

app.listen(process.env.PORT || 9999);
```

Apollo Server Setups

```
export const apolloServer = new ApolloServer({
  introspection: true,
  schema: makeExecutableSchema({
    typeDefs,
    resolvers,
  }),
  formatError: error => {
    console.error('[SERVER]: Apollo Server Error', error.extensions);
    return error;
  },
  context: async ({ ctx }) => {
    const token = getToken(ctx);

    try {
      const session = await jwt.verify(token);

      return {
        ...models,
        session,
      };
    } catch {
      return {
        ...models,
        session: null,
      };
    }
  },
});
```

Models

```
import mongoose, { Schema } from 'mongoose';
import { UUID } from 'src/models/common/UUID';

const WordSchema = new Schema({
  uuid: UUID,
  createdAt: {
    type: Date,
    default: new Date(),
  },
  translations: [
    {
      language: String,
      value: String,
    },
  ],
  stack: { type: Schema.Types.ObjectId, ref: 'Stack' },
  author: { type: Schema.Types.ObjectId, ref: 'User' },
});

export default mongoose.model('Word', WordSchema);
```

```
import mongoose, { Schema } from 'mongoose';
import { UUID } from 'src/models/common/UUID';

const StackSchema = new Schema({
  uuid: UUID,
  title: String,
  subTitle: String,
  createdAt: {
    type: Date,
    default: new Date(),
  },
  words: [{ type: Schema.Types.ObjectId, ref: 'Word' }],
  author: { type: Schema.Types.ObjectId, ref: 'User' },
});

export default mongoose.model('Stack', StackSchema);
```

Models

```
import mongoose, { Schema } from 'mongoose';
import { UUID } from 'src/models/common/UUID';

const WordSchema = new Schema({
  uuid: UUID,
  createdAt: {
    type: Date,
    default: new Date(),
  },
  translations: [
    {
      language: String,
      value: String,
    },
  ],
  stack: { type: Schema.Types.ObjectId, ref: 'Stack' },
  author: { type: Schema.Types.ObjectId, ref: 'User' },
});

export default mongoose.model('Word', WordSchema);
```

```
import mongoose, { Schema } from 'mongoose';
import { UUID } from 'src/models/common/UUID';

const StackSchema = new Schema({
  uuid: UUID,
  title: String,
  subTitle: String,
  createdAt: {
    type: Date,
    default: new Date(),
  },
  words: [{ type: Schema.Types.ObjectId, ref: 'Word' }],
  author: { type: Schema.Types.ObjectId, ref: 'User' },
});

export default mongoose.model('Stack', StackSchema);
```

Resolvers - Queries

```
import mongoose from 'mongoose';
import { getCriteria } from 'src/utils';
import { cursorOutput } from 'src/functions';

export const getWordsByStackId = async (_, { id, after, limit = 10 }, { session, Word }) => {
  const params = {
    stack: new mongoose.Types.ObjectId(id),
    author: new mongoose.Types.ObjectId(session.id),
  };

  const totalCount = await Word.find(params).countDocuments();
  const items = await Word.find({
    ...getCriteria(after),
    ...params,
  })
    .sort({ createdAt: -1 })
    .limit(limit);

  return cursorOutput(totalCount, items, limit);
};
```

Resolvers - Mutations

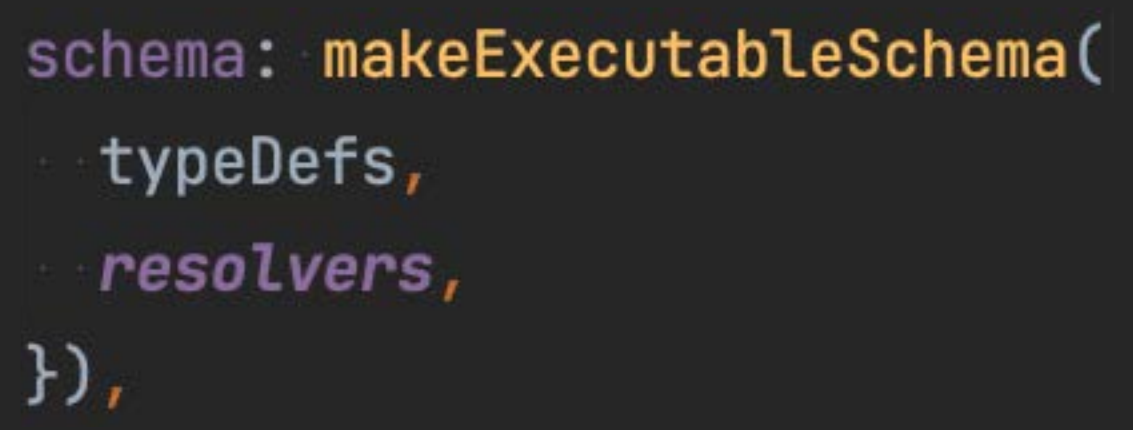
```javascript
import { getStatistic } from 'src/resolvers/Query/getStatistic';

export const updateStatistic = async (_, { mistakes = 0 }, context) => {
  const { session, User } = context;

  const user = await User.findById(session.id);
  await User.findByIdAndUpdate(session.id, {
    mistakes: user.mistakes + mistakes,
    finished: user.finished + 1,
  });

  return getStatistic(_, { period: null }, context);
};
```

Schema GraphQL

```graphql
type Query {
  getStacks(after: String, limit: Int): Stacks!
  getStack(id: String!): Stack
  getStackByUuid(uuid: String!): Stack
  getSession: User
  getStatistic(period: Period): Statistic!
}

type Mutation {
  signUp(email: String!, password: String!, firstName: String!, lastName: String!): Token!
  signIn(password: String!, email: String!): Token!
  updateUser(updateUserInput: UpdateUserInput!): User!
  updateUserPassword(newPassword: String!): User!
  updateStatistic(mistakes: Int!): Statistic!
  deleteUser: String
  deleteStack(id: String): String
  createStack(title: String!, words: [WordInput]!): Stack
  updateStack(id: String!, title: String, words: [WordInput!]): Stack
}
```

```graphql
type User {
  id: ID
  uuid: String!
  email: String!
  password: String!
  firstName: String!
  lastName: String!
  fullName: String!
  finished: Int!
  mistakes: Int!
  createdAt: DateTime!
  recoverSessionToken: String
  systemLanguage: String!
  identityProvider: String
}
```

Execute Schema with Resolvers



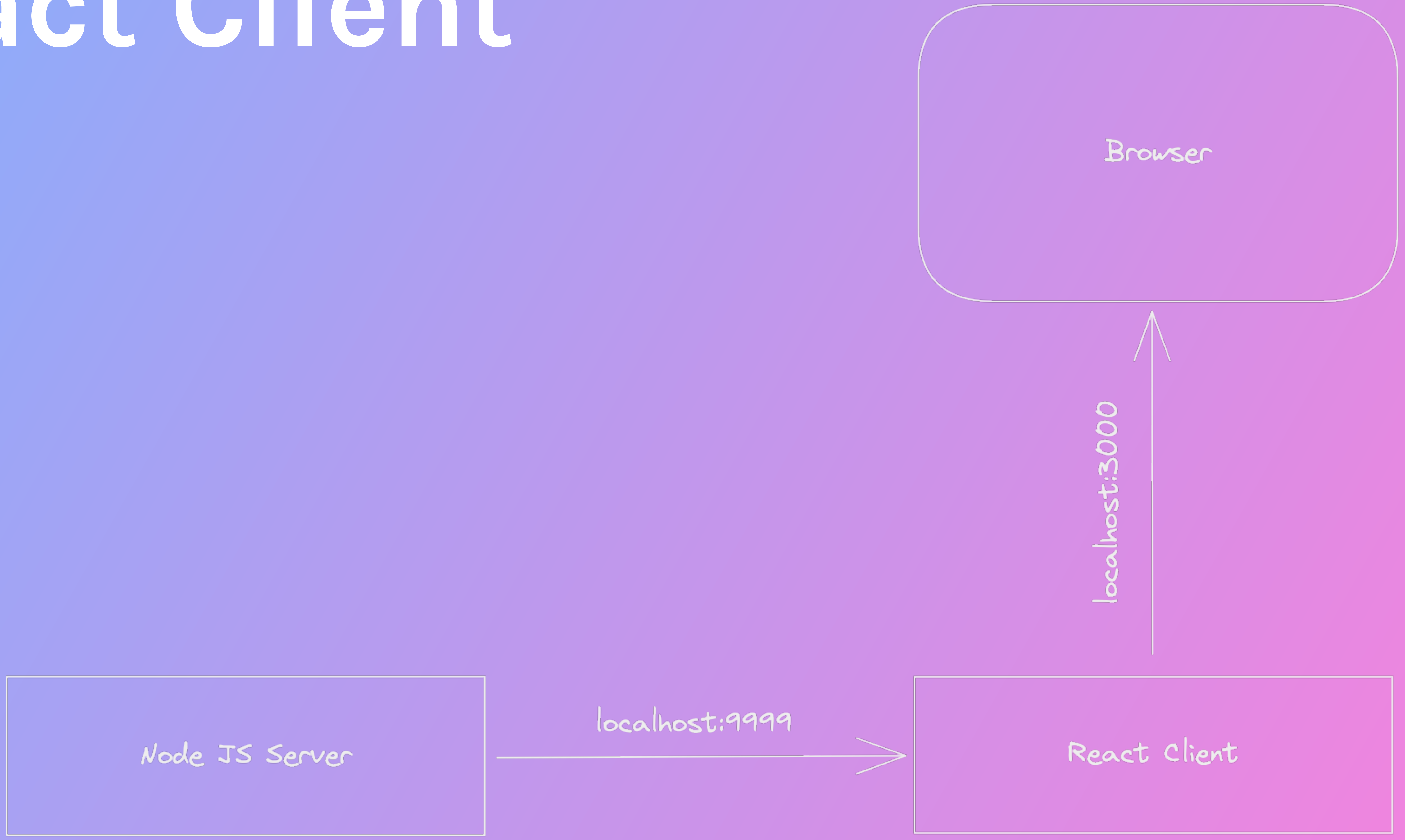
```
schema: makeExecutableSchema(
  typeDefs,
  resolvers,
}),
```

Go to Client

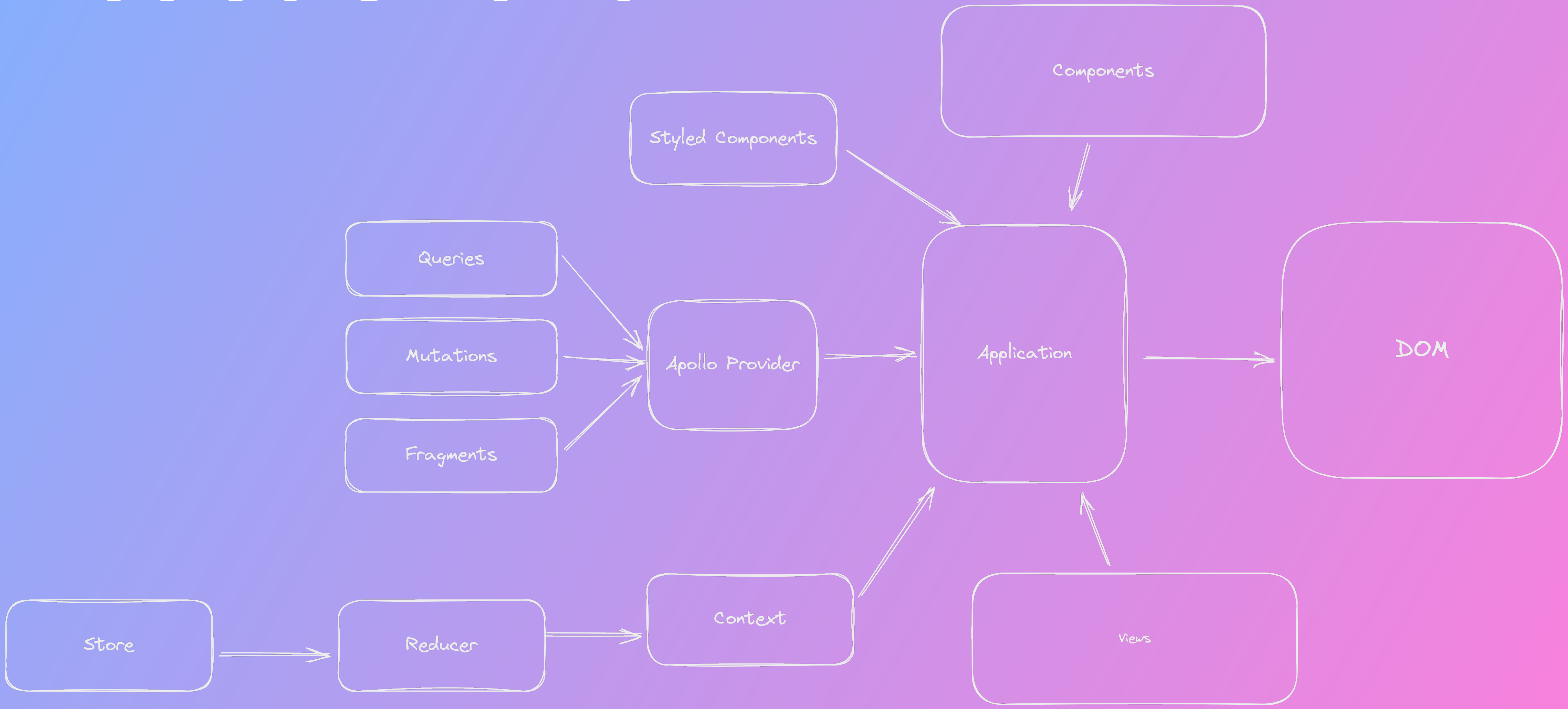
-------------------------->

localhost:9999/api/v1/graphql

React Client



React Client



Client Providers

```jsx
<React.StrictMode>
  <ApolloProvider client={apolloClient}>
    <ContextProvider>
      <ThemeProvider theme={theme}>
        <GlobalStyles />
        <App />
      </ThemeProvider>
    </ContextProvider>
  </ApolloProvider>
</React.StrictMode>,
```

Apollo Client

```
import {
  ApolloClient,
  from,
  InMemoryCache,
  createHttpLink,
} from '@apollo/client';
import { relayStylePagination } from '@apollo/client/utilities';
import { onError } from '@apollo/client/link/error';
import { setContext } from '@apollo/client/link/context';
import * as Cookies from 'es-cookie';
import { SESSION_TOKEN } from 'src/constants';

const httpLink = createHttpLink({
  uri: '/api/v1/graphql',
});
```

Apollo Error Link

```
const errorLink = onError(
  ({ graphQLErrors, networkError, operation, forward }) => {
    if (graphQLErrors) {
      graphQLErrors.map(graphQLError => {
        console.log(`[GraphQL error]: Message: ${graphQLError.message}`);
      });
    }

    if (networkError) {
      console.log(`[Network error]: ${networkError}`);
    }

    forward(operation);
  },
);
```

Auth

```
const authLink = setContext((_, { headers }) => {
  const token = Cookies.get(SESSION_TOKEN);

  return {
    headers: {
      ...headers,
      authorization: token ? `Bearer ${token}` : '',
    },
  };
});
```

Apollo Cache

```
const cache = new InMemoryCache({
  addTypename: true,
  typePolicies: {
    Query: {
      fields: {
        getAllStacksByCurrentStage: relayStylePagination(),
        getAllWordsByCursor: relayStylePagination(),
      },
    },
  },
});
```

Apollo Client with all links

```
export const apolloClient = new ApolloClient({
  link: from([errorLink, authLink, httpLink]),
  cache,
});
```

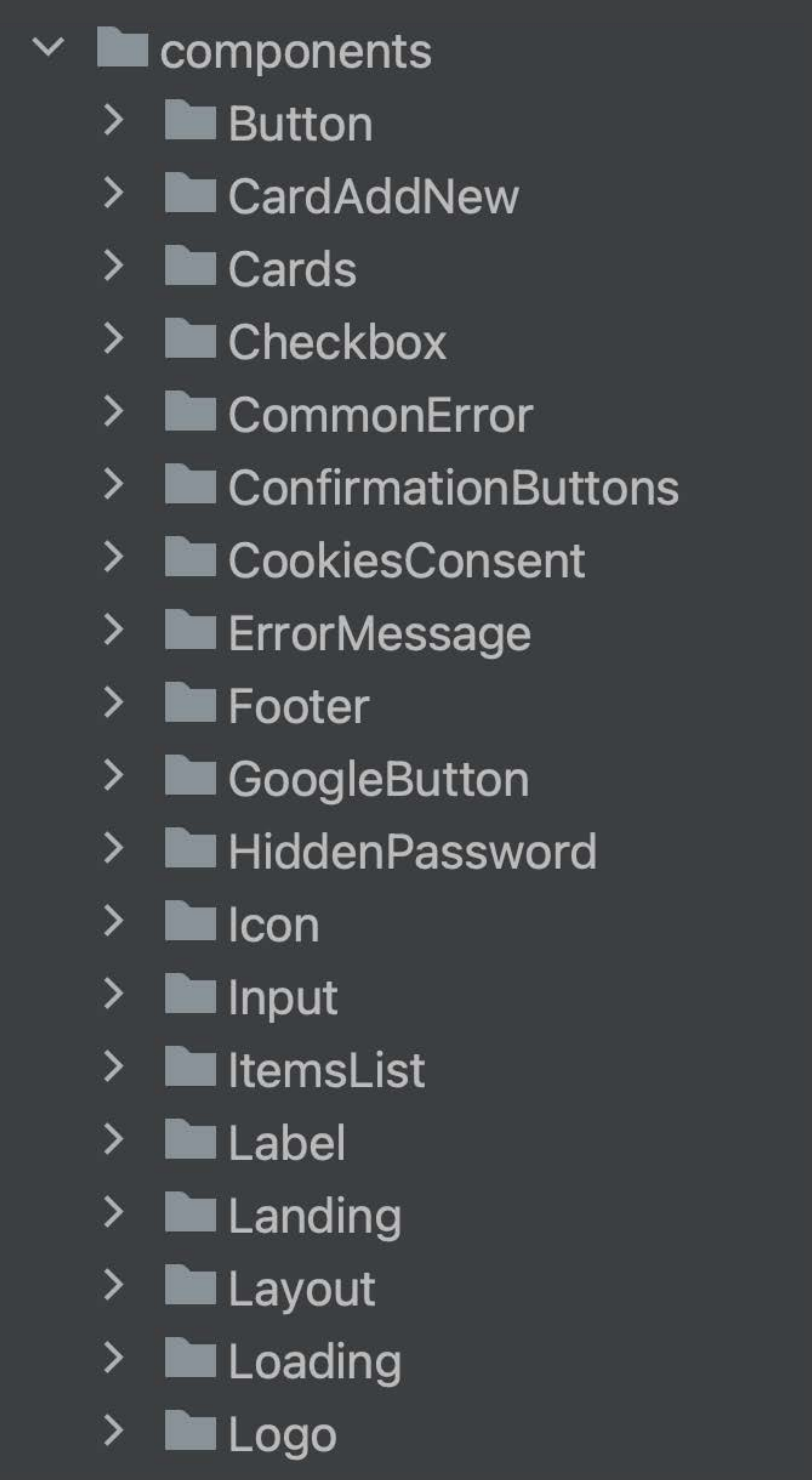
Context

```jsx
export const ContextProvider = ({ children }) => {
  const [store, dispatch] = React.useReducer(Reducer, initialState);

  return (
    <AppContext.Provider value={{ store, dispatch }}>
      {children}
    </AppContext.Provider>
  );
};

export const AppContext = React.createContext({
  store: initialState,
  dispatch: null,
});
```

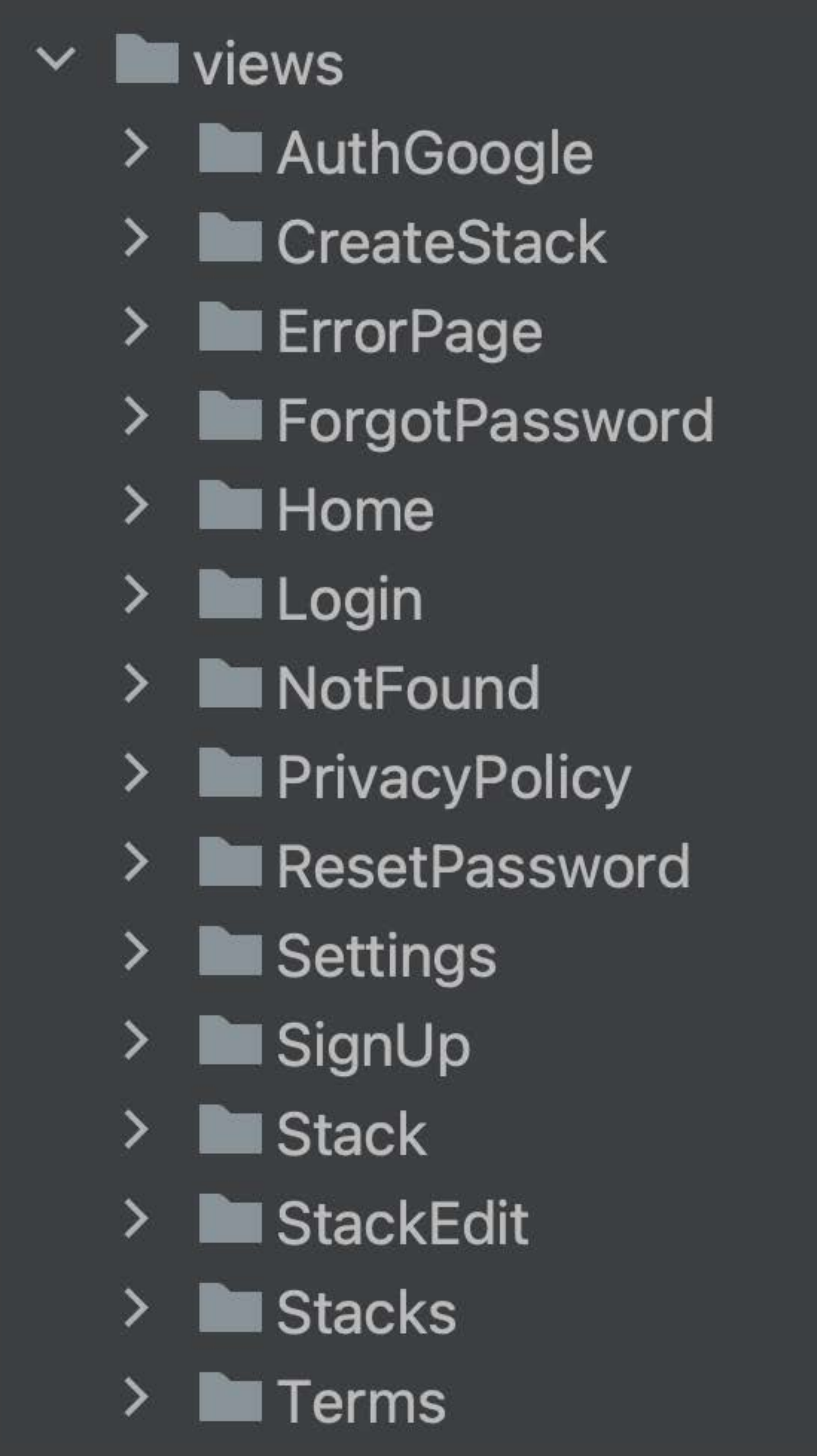
<Button />

```
Button/
  index.jsx
  Button.test.js
  styles.js
```

Views / Components



```
⌄ components
  › Button
  › CardAddNew
  › Cards
  › Checkbox
  › CommonError
  › ConfirmationButtons
  › CookiesConsent
  › ErrorMessage
  › Footer
  › GoogleButton
  › HiddenPassword
  › Icon
  › Input
  › ItemsList
  › Label
  › Landing
  › Layout
  › Loading
  › Logo
```

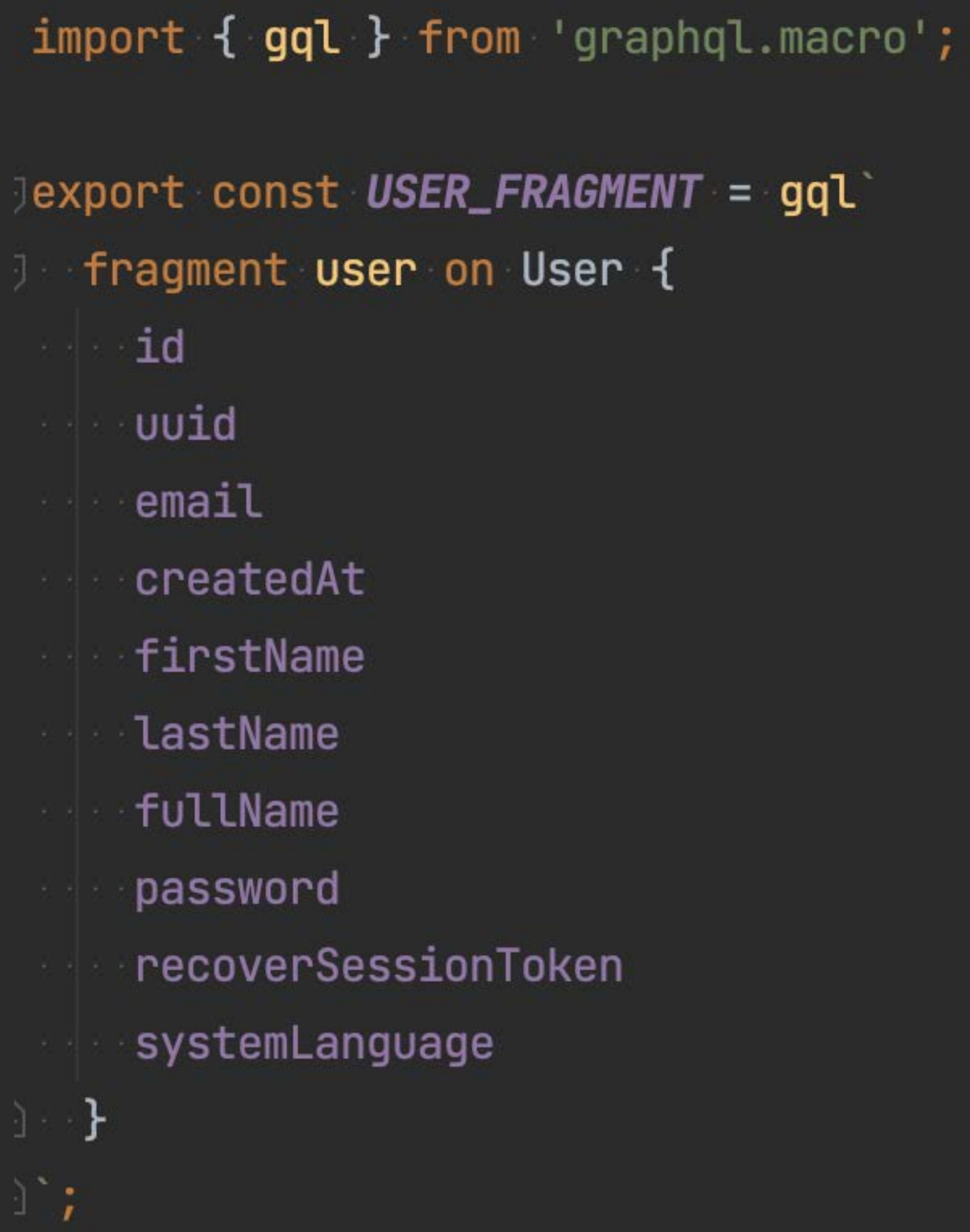


```
⌄ views
  › AuthGoogle
  › CreateStack
  › ErrorPage
  › ForgotPassword
  › Home
  › Login
  › NotFound
  › PrivacyPolicy
  › ResetPassword
  › Settings
  › SignUp
  › Stack
  › StackEdit
  › Stacks
  › Terms
```

Component

```jsx
import React from 'react';
import { ErrorMessage } from 'src/components/ErrorMessage';
import { Label } from 'src/components/Label';
import { capitalize } from 'src/utils';
import { StyledInput, StyledInputContainer } from './styles';

export const Input = ({
  name,
  value,
  error,
  onChange,
  label,
  placeholder,
  type,
  disabled,
}) => {
  const inputRef = React.useRef(null);

  return (
    <StyledInputContainer className="input">
      {label && <Label>{label}</Label>}
      <StyledInput
        placeholder={placeholder || capitalize(label)}
        error={error}
        name={name}
        value={value}
        disabled={disabled}
        onChange={onChange}
        type={type}
        ref={inputRef}
      />
      {error && <ErrorMessage message={error} />}
    </StyledInputContainer>
  );
};
```

Graphql Fragments



```
import { gql } from 'graphql.macro';

export const USER_FRAGMENT = gql`
  fragment user on User {
    id
    uuid
    email
    createdAt
    firstName
    lastName
    fullName
    password
    recoverSessionToken
    systemLanguage
  }
`;
```

GraphQL Queries

```
import { gql } from 'graphql.macro';
import { USER_FRAGMENT } from 'src/apollo/graphql/fragments';

export const GET_SESSION = gql`
  query GET_SESSION {
    getSession {
      ...user
    }
  }
  ${USER_FRAGMENT}
`;
```

useQuery

GraphQL Mutations

```
import { gql } from 'graphql.macro';
import { USER_FRAGMENT } from '../fragments';

export const SIGN_UP_USER = gql`
  mutation SIGN_UP_USER(
    $email: String!
    $password: String!
    $firstName: String!
    $lastName: String!
  ) {
    signUp(
      email: $email
      password: $password
      firstName: $firstName
      lastName: $lastName
    ) {
      token
      uuid
    }
  }
`;
```

useMutation

What to take?

How to keep all in order?

JWT Graphql Apollo Server Apollo Client SSN

AWS Security Styled Components React Router

YUP NGINX Jest Circle CI Mongo DB React

AWS EC2 Mongoose AWS Router 53 KOA

Docker Hosting Nodemailer PM2 bcrypt

EC2 - 50$

Hosting - 10$

Router53 - 5$

Thank you!