

UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Maria Belinska

Web application for managing Dag University

Bachelor's Thesis (9 ECTS)

Supervisors: Helle Hein, PhD
Taavi Sangel, Stagnation Laboratory

Tartu 2018

Web application for managing Dag University

Abstract:

The result of this thesis is an administrative application Dag Admin for Dag University. Dag University is a learning platform about cryptocurrencies, sales, marketing and personal growth. Dag Admin enables to manage content displayed in Dag University – courses, events, news, downloads and notifications. Also, some additional functionality like displaying events and downloads was added to Dag University. During the development, React, Node.js, GraphQL, TypeScript and MariaDB were used.

Keywords:

Web application, Dag University, Dag Admin, React, GraphQL, Node.js, TypeScript, MariaDB

CERCS: P170

Veebirakendus Dag University haldamiseks

Lühikokkuvõte:

Käesoleva töö raames arendati administreerimise rakendust Dag Admin portaalile Dag University. Dag University on õppeplatvorm krüptovaluuta, müügi ja turunduse õppimisele ning üldisele iseenda arendamisele. Dag Admin võimaldab hallata sisu, mis on nähtav Dag University lehel – kursuseid, üritusi, uudiseid, lisasid ning teavitusi. Samuti lisati uut funktsionaalsust nagu ürituste ja lisade kuvamine portaalile Dag University. Arenduses kasutati tehnoloogiaid React, Node.js, GraphQL, TypeScript ning MariaDB.

Võtmesõnad:

Veebirakendus, Dag University, Dag Admin, React, GraphQL, Node.js, TypeScript, MariaDB

CERCS: P170

Table of Contents

1. Introduction	4
2. Comparison with existing solutions	5
3. Technologies	6
3.1 React	6
3.2 GraphQL	6
3.3 TypeScript	7
3.4 Node.js	8
3.5 MariaDB	9
4. Application architecture and design	10
4.1 Functional requirements	10
4.2 Non-functional requirements	11
4.3 Client-side solution	11
Managing content	11
Verification system	13
Managing translations files	14
Logging	15
Displaying content in Dag University	15
4.4 Server-side solution	17
Schema	17
Resolver	18
Queries	18
Middlewares	19
Services	19
4.5 Database model	20
5. Conclusion and future work	21
6. References	22
Appendix	23
I. Source code	23
II. License	24

1. Introduction

Today's society is constantly changing and requiring new skills to be successful. The problem is that schools do not always keep up with the flow and tend to become outdated. Dag University has been developed to solve some of the shortcomings of traditional school systems. For example, many subjects taught at schools do not contain important fields like sales, digital marketing, leadership and business owning. Or if they do, students are not provided opportunities to use these skills to make an income. Dag University covers these needs to get as useful education as possible. In addition, to make it more flexible for students, the courses can be accessed from anywhere, anytime and any device. Because people have different learning styles, Dag University is providing different types of materials like videos, audio books, PDF-books and different problems to solve. [1]

The problem was that for some modules there was no administration in Dag University. The purpose of this thesis was to add functionality to manage events, news, downloads, FAQs, verifications, languages, payments and translation files in Dag Admin. During the work, it was also necessary to change some code in Dag University so that the content displayed would be taken from database.

The application was built considering modern and innovative technologies. For example, in front-end development, React was used – a JavaScript library for fast rendering with virtual DOM [2] and creating reusable components [3]. It was combined with TypeScript, which allows to add static type annotations and therefore makes code less error-prone [4]. In the backend, GraphQL and Node.js were used. GraphQL is an efficient data query language for sending data from server to client [5]. Node.js is fast and scalable [6] environment for making server-side applications. For database management, MariaDB was used – a powerful and easy to use database [7], which has many performance improvements and optimizations compared to MySQL [8].

The thesis also gives a detailed overview of Dag Admin application's architecture and design. Client-side code consists of list and detail views. The API endpoints consist of schema, resolver and queries. In the schema, GraphQL types are defined, resolver handles the data that has to be changed to some other form and queries make SQL-statements to get the data from the database. Middlewares are for getting certain files when user hits a specific URL. Services are made to avoid the duplication of code and to access these functions throughout the project. There are also used various Node.js libraries, for example for finding differences between objects or for paginating pages. The reason to use third-party libraries is that it makes development cycle more convenient and faster, because there is no need to write the code that has already been created by somebody else.

The thesis is organized as follows: Chapter 2 provides a comparison with existing solutions and points out what are the strengths of Dag University. Chapter 3 gives a detailed overview of used technologies. It describes why they are used, what are their advantages and where are the drawbacks. Chapter 4 describes application's architecture and design. There are screenshots of final applications as well as explanation how the backend works. Chapter 5 summarizes what has been done during the work and offers new opportunities to continue the project.

2. Comparison with existing solutions

Dag University is specialized on providing content about cryptocurrencies, sales and digital marketing. The subjects are combined, materials are constantly updated and created by top experts in their fields. In addition, there are opportunities to use different learning methods like watching videos, reading PDFs, listening to mp3's and solving different exercises. However, there are also other websites that offer courses on the same subjects. Some of them are Udemy, Coursera, Coincademy and LeanCryptography.

One special feature in Dag University is an opportunity for team building. Users have the possibility to build their own team of sales people and start making income after short amount of time. Also, in Dag University, it is possible to trade new and fast cryptocurrency dagcoin, but also use other common cryptocurrencies like bitcoin. Moreover, Dag University offers free dagcoins for all students who pass the courses. It is also possible to become an affiliate and start earning commissions from 10% and up [1].

Dag University stands out for constantly organizing various events in different countries for the students to meet, get to know each other and to learn more hands-on about sales and cryptocurrencies.

3. Technologies

3.1 React

React is an open-source JavaScript library [3] for front-end development that enables to build complex user interfaces [9]. It is fast, simple and predictable [2]. React communicates with data mainly through props and state. Props enable to pass information through attributes from parent to child. State is used to hold variables, which value can change within the component. Each time the state is changed, the parts of page that have changed are reloaded. This is one of the most convenient feature for developers – there is no need to worry about data changing - every time a state is changed, the changes are displayed immediately.

In React, it is convenient to create reusable components. The reason why to use them is that web pages often consist of similar pieces that tend to repeat. If some logic in page changes, developer has to change code only in one place. It also makes development much faster and less error-prone. Although each developer can create its own components, React libraries provide various premade components.

Components main function is `render()` that contains information about what will be displayed on the page. It is called JSX and enables to use HTML tags in JavaScript that makes writing code more convenient and understandable. The example of JSX is presented on Figure 2.1. For rendering, React uses Virtual-DOM that makes web pages much faster [2]. Good performance indicators is one of the reasons why React is so popular.

```
public render() {
  if (!this.props.data) {
    |   return <Loader />;
  }

  const { user } = this.props.data;
  const { isEditable, formRevision } = this.state;

  if (!user) {
    |   return null;
  }

  const activePacket = packageList.find(item => item.id === user.activePacketId);
  const activeRank = rankList.find(item => item.id === user.rankId);

  return (
    <div className="View UserDetailsInformationView">
      <Form onSubmit={this.handleSaveUser} key={formRevision} promptContent={this.renderSaveUserPrompt}>
        <Columns>
          <Panel title="Main information">
            <Field name="firstName" label="First name" defaultValue={user.firstName} disabled={!isEditable} />
          </Panel>
        </Columns>
      </Form>
    </div>
  );
}
```

Figure 2.1 Example of JSX that enables to use HTML tags in JavaScript.

3.2 GraphQL

GraphQL is a data query language that is used for sending data from server to client. It is open-source, efficient and easy to use. [5]

GraphQL was build as an alternative to REST (Representational State Transfer). There are several aspects where GraphQL is better than REST. For example, there is no problem with over- or under-fetching [5], because it is possible to ask only for the data that is needed. It also eliminates some security risks, because the client does not get all the data. Moreover, there is less performance issues, because there is no overloading. The example of retrieving data with GraphQL is shown on Figure 2.2.

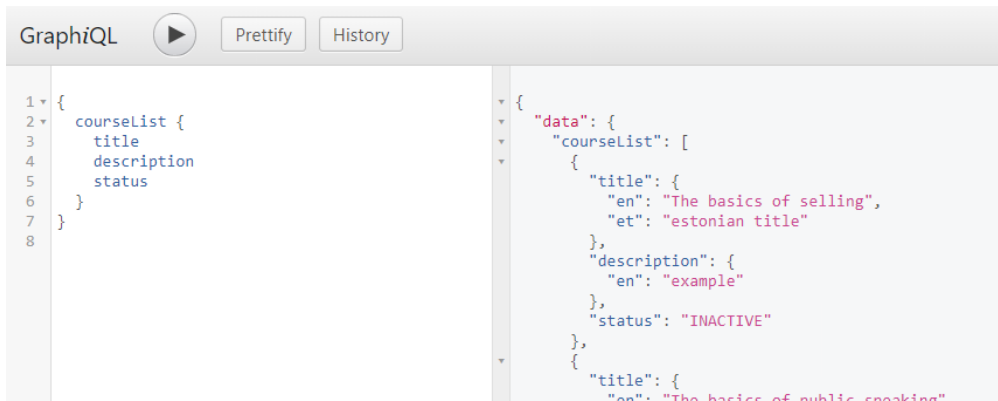


Figure 2.2. Retrieving data with GraphQL.

Another remarkable advantage of GraphQL is that there is no need to manually write documentations, it can be auto-generated from schemas. GraphQL schemas are strongly typed and editors give autocompletes when writing a query. In REST, on the other hand, working with different APIs can be unpredictable, because developers use different specifications, there are no common standards and documentations tend to become outdated [5].

In GraphQL schemas, there are defined Queries and Mutations. Queries are for asking data and Mutations for changing data. All of them must have a return value type and developers can also add their own types. It is also possible to add enum and input types, parameters and determine whether a certain field is mandatory or not.

3.3 TypeScript

TypeScript is a superset of JavaScript, which means that all valid JavaScript is also valid TypeScript [4]. Because of that, it takes a little time to learn and start using TypeScript. It is open source and runs on every browser and OS [10]. The comparison between TypeScript and JavaScript is shown on Figure 2.3.

Compared to JavaScript, TypeScript helps to find errors before running the code [10] and saves a lot of time because less debugging is needed. The most common errors that TypeScript finds is that object is undefined, object does not contain certain attributes or the types are incompatible.

TypeScript becomes very useful when refactoring code. During the development, Dag Admin was refactored repeatedly – it means that some parts of code had to be improved or

changed. TypeScript shows immediately where code becomes broken and does not compile until errors are fixed.

Typescript	Javascript
<pre>class Greeter { greeting: string; constructor (message: string) { this.greeting = message; } greet() { return "Hello, " + this.greeting; } }</pre>	<pre>var Greeter = (function () { function Greeter(message) { this.greeting = message; } Greeter.prototype.greet = function () { return "Hello, " + this.greeting; }; return Greeter; })();</pre>

Figure 2.3. Comparison between TypeScript and JavaScript [11].

Although TypeScript makes in general code writing easier, it can also be time consuming to understand it in the beginning. Also, when using Node.js, the libraries used must be type annotated. Although many of these annotations can be installed, developers must often write these themselves. However, once programmer is used to TypeScript, it speeds the development up and is especially useful against typing mistakes. It makes code safer [10] and shorter.

TypeScript is suitable for this project because the codebase is getting larger with time and TypeScript makes it easier to manage.

3.4 Node.js

Node.js is an open-source environment for building server-side applications and is used by many large companies. Node.js uses asynchronous programming, which means that several operations occur in parallel. It is efficient regarding memory and time. [6]

One benefit of Node.js is that the same language can be used both in client and in server. It means that some code can be shared and there is less switching as it would be with using Java or Ruby in the backend.

Node.js community is active and constantly growing. It gives an opportunity to choose between many libraries that can be installed with NPM (Node.js Package Manager). It is recommended to use them and write as few code as possible, because it reduces the probability of writing faulty code. Node.js and NPM make a good platform for developing high-performance applications. [12]

Although Node.js is a good solution for most applications, it is not meant for heavy-computing apps. In addition, because Node.js uses JavaScript, it is important to be cautious and avoid common JavaScript bad practices like using too many nested callbacks. Moreover,

although there are many good packages that can be installed with NPM, not all are maintained well which can later cause bugs that are hard to debug.

3.5 MariaDB

MariaDB is a Relational Database Management System (RDBMS) that is fully open source. It is a fork of MySQL and was started by some core developers of MySQL. There are some new additions in MariaDB like performance improvements, optimizations, better testing and bug fixes that MySQL does not have [8].

MariaDB is for creating and managing relational databases. Like with other relational database management systems, it is possible to change the data and its structure as well as answer queries. To communicate with MariaDB, user has to write SQL statements (Structured Query Language). [13]

MariaDB has quickly gained a large number of users with past few years and today it is powering many websites and companies. It is a powerful database and very easy to install and use. [7]

4. Application architecture and design

4.1 Functional requirements

Functional requirements describe the actions that users must be able to perform within the application. The requirements are listed in Table 1.

Table 1. Functional requirements for Dag University and Dag Admin.

ID	Requirement
1	In Dag Admin, administrators should be able to view, add, edit and delete news.
2	In Dag Admin, administrators should be able to view, add, edit and delete events.
3	In Dag Admin, administrators should be able to view, add, edit and delete downloads.
4	In Dag Admin, administrators should be able to view, add, edit and delete FAQs.
5	In Dag Admin, administrators should be able to view, upload and download translation files.
6	In Dag Admin, administrators should be able to view, add, edit and delete languages.
7	In Dag Admin, administrators should be able to view and filter payments by payment id, transaction id, payment method and payment status.
8	In Dag Admin, administrators should be able to view and filter direct sales by date and amount.
9	In Dag Admin, administrators should be able to view and filter volume points by from date and to date.
10	In Dag Admin, administrator's actions should be logged. There should also be an opportunity to see these logs and filter them by date, permission and action.
11	In Dag Admin, some administrators should be able to view, add and delete other administrator accounts. They should also be able to choose which permissions each administrator has.
12	In Dag University, news that are displayed should be taken from database.
13	In Dag University, events that are displayed should be taken from database.
14	In Dag University, downloads that are displayed should be taken from database.

15	In Dag University, FAQs that are displayed should be taken from database.
----	---

4.2 Non-functional requirements

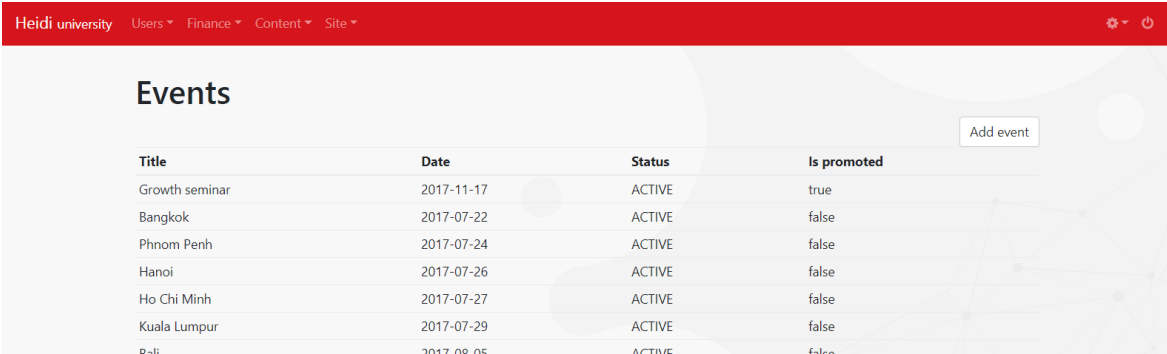
Non-functional requirements describe the limitations of the application and expectations for the users. The requirements are following:

- 1) Each page should not load more than one second.
- 2) User should be able to learn how the system works within a day.
- 3) User has to have an internet connection and web browser to use application.
- 4) Applications should run on every OS.
- 5) Dag University should run on every browser. Dag Admin's support is focused on Google Chrome.
- 6) Dag University has to be able to handle thousands of users and Dag Admin 20 users.
- 7) User's data must be protected.

4.3 Client-side solution

Managing content

There are different types of content in Dag University that need to be administrated in Dag Admin – events, news, downloads, notifications, payments and FAQs. Each module contains of two main views – list view and detail view. List view shows all the elements in the selected module. It also has a button in the top corner to add a new element. An example of list view is provided on Figure 4.3.1.



Title	Date	Status	Is promoted
Growth seminar	2017-11-17	ACTIVE	true
Bangkok	2017-07-22	ACTIVE	false
Phnom Penh	2017-07-24	ACTIVE	false
Hanoi	2017-07-26	ACTIVE	false
Ho Chi Minh	2017-07-27	ACTIVE	false
Kuala Lumpur	2017-07-29	ACTIVE	false
Dali	2017-08-05	ACTIVE	false

Figure 4.3.1. List view of events.

Detail view enables to add, edit and delete data. When clicked on 'Add' button in list view, it leads to detail view, which has blank fields that can be filled out. Some fields are already

pre-filled. It is also possible to choose the language in which the text is entered. For that, the user has to click on the flag icon and the list of all languages will be displayed (default language is English). For entering content information, there is an editor that gives an opportunity to style the text. When clicked on 'Cancel', the list view will be shown again. An example of adding new element is on Figure 4.3.2.

Figure 4.3.2. Adding new element.

In detail view, it is also possible to see and edit data of existing element. The view is quite similar to as adding a new element, except the fields are filled out. When clicked on 'Edit', all the fields become editable. There is also a button for deleting the element. Example of detail view can be seen on Figure 4.3.3.

Figure 4.3.3. Detail view of an event.

Each time when making changes and trying to save them, the confirmation modal will pop up. It shows all the changes with previous and current values. The reason is to provide some extra check for administrators to be sure that the information they entered is correct. The example of confirmation modal is on Figure 4.3.4.

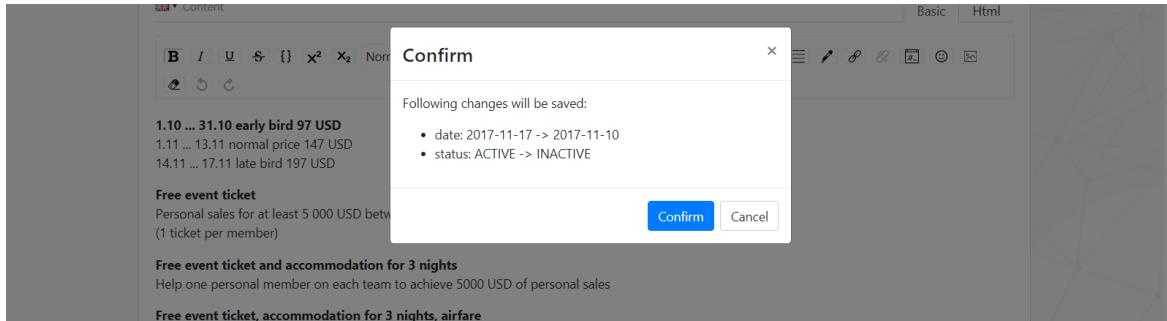


Figure 4.3.4. Confirmation modal.

There is also a need for uploading files since some modules like events and news have to display images. When some image has already been uploaded, it is possible to choose it under 'Select' tab. Example of uploading a file is on Figure 4.3.5.

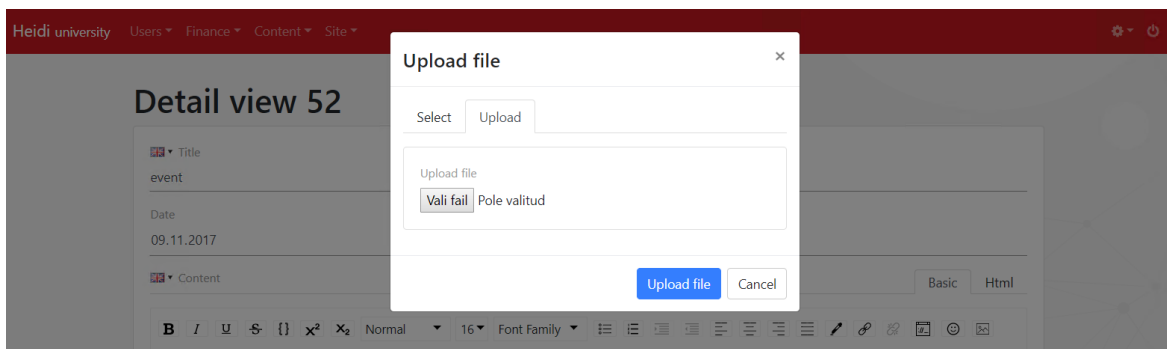


Figure 4.3.5. Uploading a file.

Verification system

Another functionality of Dag Admin is to verify users. The reason why users need to be verified is that it is possible to make payments within the Dag University's application. To be verified, Dag University users enter their information with pictures of documents and administrators have to verify that the entered fields are correct. Verification system was redone several times, so it would be as convenient as possible for the administrators, since they have to verify thousands of users. Latest version is that correct document and fields are positioned side by side and it is possible to zoom the document. Below there are thumbnails of other documents and it also shows which ones are already verified. Verification system is shown on Figure 4.3.6.

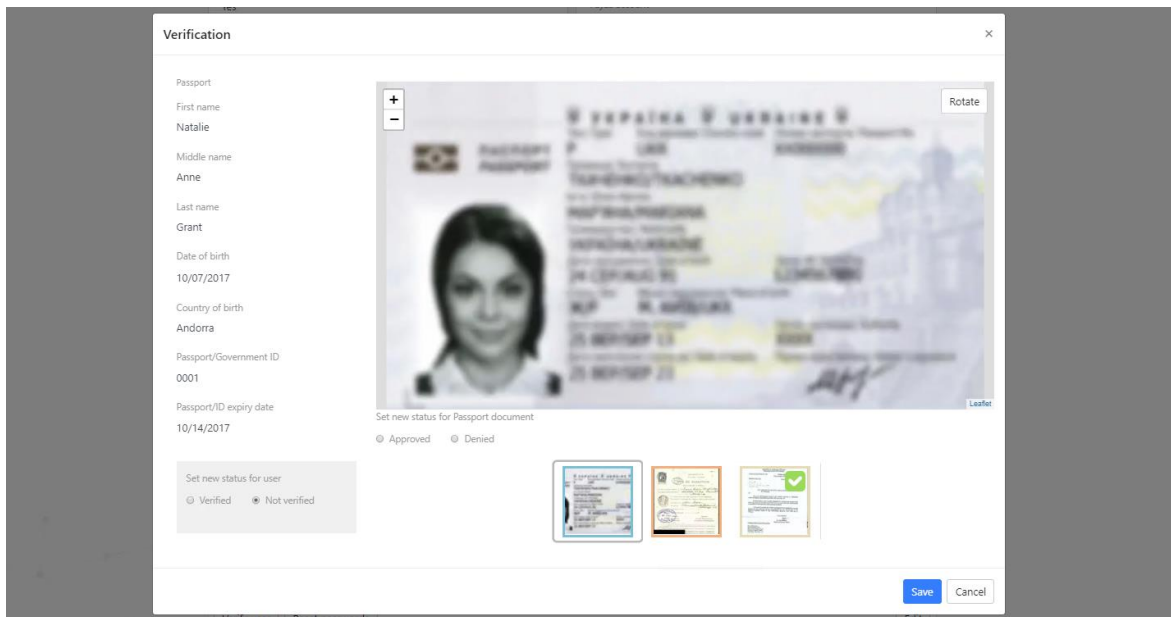


Figure 4.3.6. Verifying user.

Managing translations files

Another important aspect is that there are many different languages that can be chosen when using Dag University. For displaying static information, administrators upload translation (.po) files that Dag University will use to display text. It is also possible to add new languages and choose whether a language is currently visible or not in Dag University. Translations view also shows all translation files that have been uploaded to Dag University and it is possible to download each version and restore to some previous state. Example of translation view is on Figure 4.3.7.

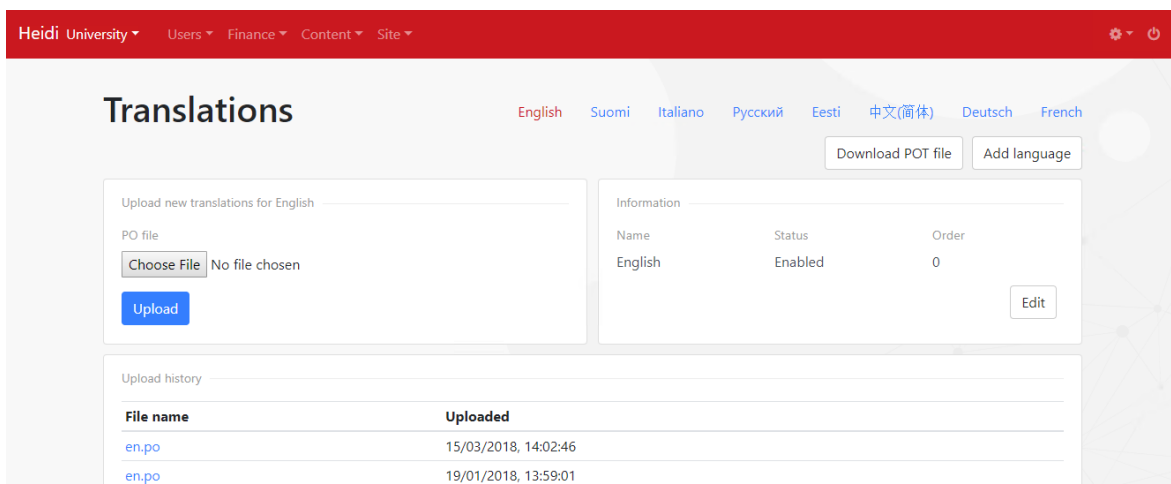


Figure 4.3.7. Translations view.

When uploading new translation, the confirmation modal will pop up with all the changes that have been made compared to the latest translation file. It shows translation keys and

below new values with green color and previous values with red color. The example is on Figure 4.3.8.



Figure 4.3.8. Showing changes when uploading a new translation file.

Logging

There was also a need to log all actions administrators make that change the database. The reason is that there would be an opportunity to track later who has made certain change or how much work an administrator has done. In log view, there is shown an action, a permission with what the action was made, id of element, date and time. It is also possible to filter logs by different parameters like date, permission or choose how many items should be per page. Log view is shown on Figure 4.3.9.

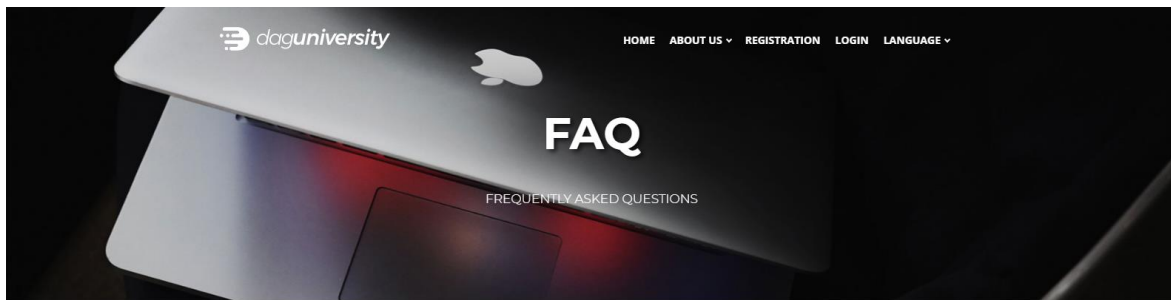
Action	Permission	Element id	Date	Time
Update news info	Administer news	1	15/03/2018	18:57:30
Update event info	Administer events	52	15/03/2018	17:59:56
Update news info	Administer news	36	15/03/2018	17:56:45
Update news info	Administer news	1	15/03/2018	17:56:12
Update news info	Administer news	1	15/03/2018	17:55:21
Update news info	Administer news	1	15/03/2018	17:55:06
Add translation file	Administer translations	117	15/03/2018	17:54:51

Figure 4.3.9. Log view of administrator.

Displaying content in Dag University

The modules that are editable in Dag Admin must also be viewable in Dag University. There are two main views – public and private. Public views are accessible for everyone, private views are only for those who have an account and are logged in.

FAQs are both in public and private view. They consist of commonly asked questions and their answers. The examples are on Figure 4.3.10 and Figure 4.3.11.



Q: What is DagUniversity?

DagUniversity is an online education platform. We believe that every modern school should teach about the newest technologies, new possibilities and show how we all can benefit from them. And this is exactly why we created DagUniversity.

Q: Are there any additional costs, besides the cost of the course?

No, just choose and purchase your favorite courses. There are no monthly or yearly subscription fees or any other hidden fees.

Figure 4.3.10. FAQs in Dag University's public view.

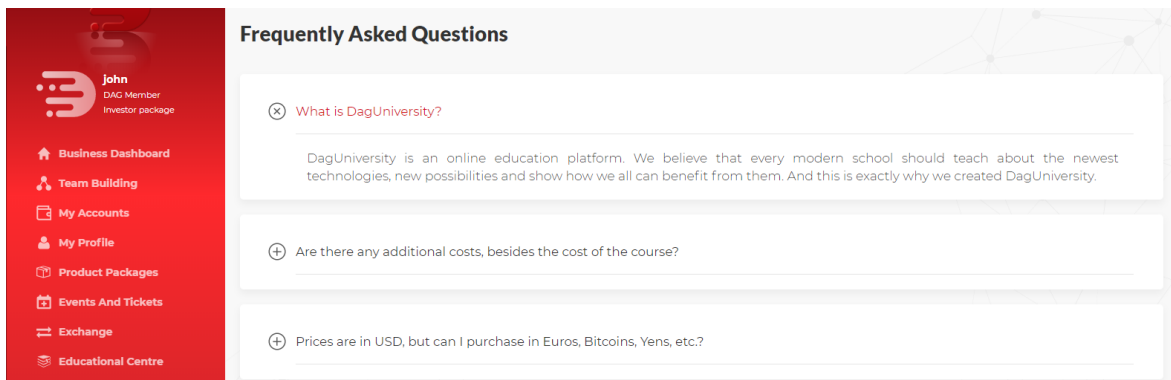


Figure 4.3.11. FAQs in Dag University's private view.

Events are only seen when logged in. The view has three different sections – promoted events, upcoming events and past events. For each event, there is a poster, name and date. Events are shown on Figure 4.3.12.

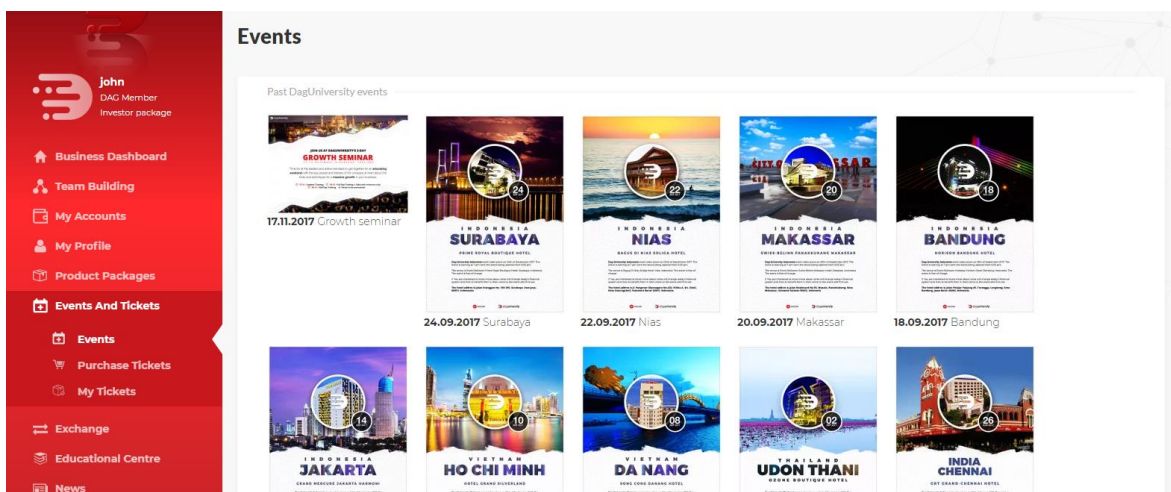


Figure 4.3.12. Events view in Dag University.

In Dag University, it is also possible to see downloads that are grouped by different categories. These are a list of files that can be useful for the users. The example is on Figure 4.3.13.

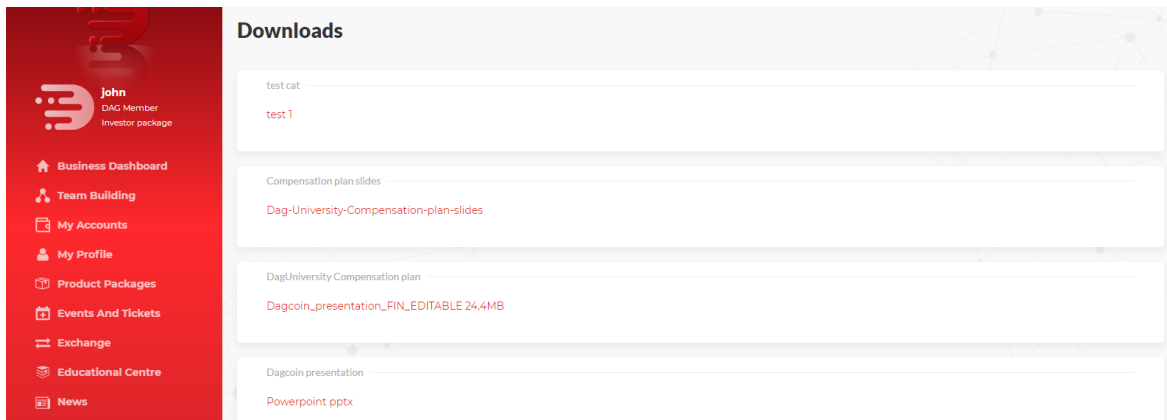


Figure 4.3.13. Downloads view in Dag University.

4.4 Server-side solution

Server-side code consists of schemas, resolvers, queries, middlewares and services.

Schema is a *.gql*-type file that stands for GraphQL. It is a type-declaration file where type-annotation are written for type-safety – it decreases the amount of errors found during runtime. There can be used both existing basic types like String, Int, Boolean as well as self-written types. Exclamation mark stands for mandatory field, which means that object always has it. The example of schema file is shown on Figure 4.4.1.

```
type Query {  
  |  eventList: [Event]!  
  |  event(id: ID!): Event  
  |  
}  
  
type Mutation {  
  |  addEvent(input: AddEventInput!): AddEventResult  
  |  updateEventInfo(id: ID!, input: UpdateEventInfoInput!): UpdateEventInfoResult  
  |  deleteEvent(id: ID!): DeleteEventResult  
  |  
}  
  
type Event {  
  |  id: ID!  
  |  title: Translation!  
  |  content: Translation
```

Figure 4.4.1. Example of schema file.

Schema also determines the structure of the data that is going to be received. Main types are Query and Mutation. Query is like a GET-method – used for fetching data and Mutation is like a POST-method – used for changing data.

Great advantage of using GraphQL and not REST technologies is that it allows to ask only for the data that is needed – it makes it more secure, since not all data can be reached.

Resolver is a *.ts*-file, which stands for TypeScript that enables to make code as type-safe as a code-writer wants. Resolver is a place to handle objects that need to be solved differently – each object, query and mutation can be resolved individually. The example of a resolver is shown on Figure 4.4.2.

In current project, sometimes data needs to be parsed to some other form to be readable in the database. Also, in resolver, all the permissions are checked, so that users would not have access to views or actions they are not allowed to have. Moreover, resolver validates the input, for example, if the password is strong enough or if the user exists in database. In addition, it can transform data into desired shape, for example, to JSON.

```
export default {
  Event: {
    title: data => (data.title ? JSON.parse(data.title) : data.title),
    content: data => (data.content ? JSON.parse(data.content) : data.content),
    isPromoted: data => data.isPromoted === '1',
    hasTickets: data => data.hasTickets === '1',
  },

  Query: {
    eventList: withPermission('DAG_CONTENT_EVENTS_VIEW', eventQueries.getEventList),
    event: withPermission('DAG_CONTENT_EVENTS_VIEW', eventQueries.getEventById),
  },

  Mutation: {
    addEvent: withPermission(
      'DAG_CONTENT_EVENTS_EDIT',
      (parent args: AddEventMutationArgs ctx: Info) =>
```

Figure 4.4.2. Example of resolver file.

Queries is a *.ts*-file. It consists of various queries – SQL-statements. It communicates with the database – asks for information and changes it. There are also used different additional functions that make fetching from database quicker and safer. The example of queries is shown on Figure 4.4.3.

Typical queries are for asking certain element by its id or by some other additional parameters. Also, sometimes there are queries asking for all the rows that are in the table. As for mutations, there are queries for adding, updating and deleting elements. Tables are often joined, when getting information depends on several tables.

```

export const getEventList = (parent, args, ctx, info) =>
  ctx.db.queryNested(`SELECT ${from('events', 'id', info, joinEventTables)} WHERE events.status != "DELETED"`, args);

export const getEventById = (parent, args, ctx, info) =>
  ctx.db
    .queryNested(
      `SELECT ${from('events', 'id', info, joinEventTables)} WHERE events.id = :id AND events.status != "DELETED"`,
      args,
    )
    .then(res => res[0]);

export const updateEventInfo = (parent, { id, input }, ctx, info) =>
  ctx.db.query(update('events', id, input), { id, ...input }).then(res => res.info);

export const deleteEvent = (parent, { id }, ctx, info) =>

```

Figure 4.4.3. Example of queries file.

Middlewares get in action when user hits a certain URL that is served by some middleware. It uses an Express Router to get and send HTTP requests. One way of using middleware is, for example, for fetching files. There is a file middleware that has an endpoint `/files/:id`. The middleware searches for the file with entered ID, converts it into a buffer, writes content type and length to header and sends it to the server. The advantage of converting to buffer and specifying data length is that the browser can already start loading and showing some parts of the file without user having to wait too long until the whole file is loaded. Therefore, it creates a better user experience.

Middlewares are also used to download files or authenticate users. The example of middleware is shown on Figure 4.4.4.

```

if (isTranslationTemplate) {
  translation = await getLastTranslationTemplateFile(null, null, { db }, ['rawContent']);
  translation.fileName = 'translation_template.pot';
} else {
  translation = await getTranslationFileById(null, { id: req.params.id }, { db }, ['rawContent', 'fileName']);
}

if (!translation) {
  res.status(404).send('File not found!');
}

const buffer = new Buffer(translation.rawContent, 'UTF-8');

res.writeHead(200, {
  'Content-Type': 'text/plain; charset=UTF-8',
  'Content-Length': buffer.length,
  'Content-disposition': `attachment;filename*=UTF-8'${encodeURIComponent(translation.fileName)}'`,
  'Transfer-Encoding': '8bit',
});

res.end(buffer);

```

Figure 4.4.4. Example of middleware.

Services are TypeScript-files that consist of one or several functions that are used throughout the server code. This way can be avoided the duplication of code and it makes much more convenient to refactor the code.

In current project, there are, for example, services for generating random string-number combinations, checking user permissions, logging user actions, formatting date-time and currencies. Example of a service is shown on Figure 4.4.5.

```

export default function logUserAction(parent: any, args: any, ctx: any, info: any, permission: any, res: any) {
  const isMutation = info.parentType.toString() === 'Mutation';

  if (!isMutation) {
    return;
  }

  const content: { id: string } = { id: null };

  !res.insertId
    ? (content.id = res)
    : res.insertId.toString() === '0' ? (content.id = args.id) : (content.id = res.insertId);

  const mutationName = info.fieldName;
  const userId = ctx.viewer.id;

  addAdminUserLog(parent, { input: { permission, content: JSON.stringify(content), mutationName, userId } }, ctx, info

```

Figure 4.4.5. Example of service.

4.5 Database model

Below is the database model (Figure 4.5.1) for the functionalities that were created during the work. The model illustrates relationships between the tables.

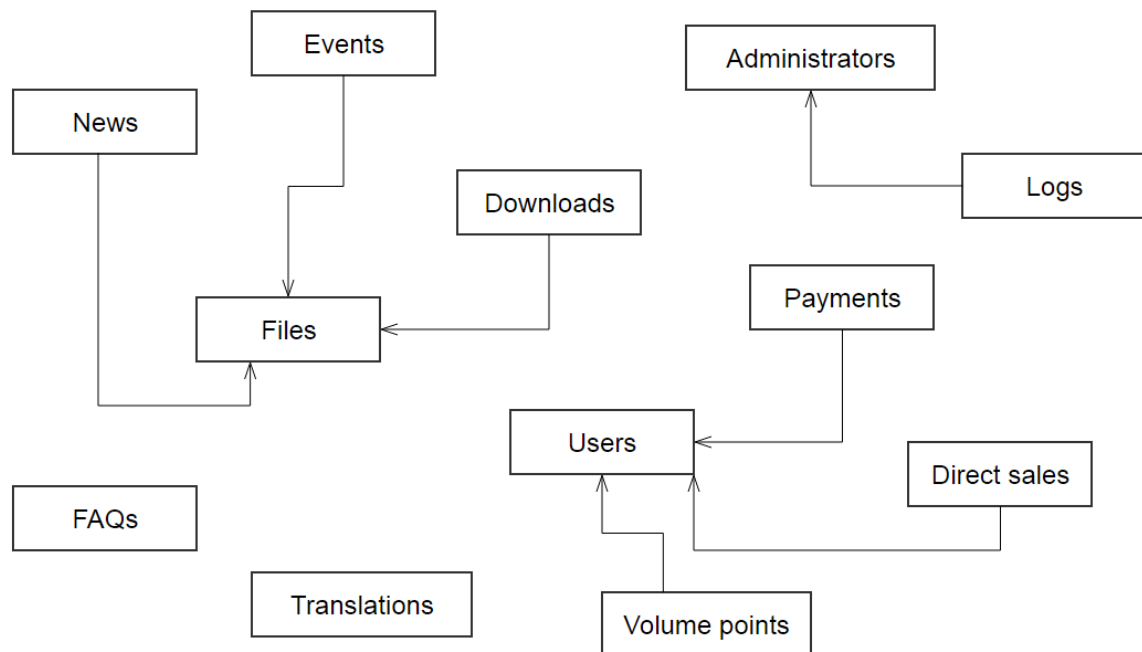


Figure 4.5.1. Database model.

There are tables Events, News and Downloads that have a file id, which references to one file in the Files table. Payments, Direct sales and Volume points are all associated with one User, therefore they have an user id column. Each log in Logs table is linked with an administrator in Administrators table. FAQs and Translations do not depend on other tables.

5. Conclusion and future work

The purpose of this thesis was to develop web application for managing Dag University. It enables to administrate events, news, downloads, notifications, FAQs and other content that is displayed in Dag University. It also has a functionality to verify users, upload translation files and log user actions.

In the process, additional functionality was added to Dag University, for example, displaying the information that can be inserted into Dag Admin, like news, events, downloads, FAQs in both public and private view.

In the future, there are plans to make Dag Admin support also other Dag web applications like Merchant Finder (for finding businesses that accept dag payments) and SwipeX (for purchasing popular cryptocurrencies), so that they could all be easily managed in one application. Main functionalities would be verifying users and companies as well as editing their information.

6. References

- [1] Dag University. <https://daguniversity.com> (25.04.18)
- [2] Fedosejev, A., 2015. React.js Essentials. Packt Publishing Ltd.
- [3] Vipul, A.M. and Sonpatki, P., 2016. ReactJS by Example-Building Modern Web Applications with React. Packt Publishing Ltd.
- [4] Jansen, R.H., 2015. Learning TypeScript. Packt Publishing Ltd.
- [5] Buna, S., 2016. Learning GraphQL and Relay. Packt Publishing Ltd.
- [6] Teixeira, P., 2012. Professional Node.js: Building Javascript based scalable software. John Wiley & Sons.
- [7] Bartholomew, D., 2013. Getting Started with MariaDB. Packt Publishing Ltd.
- [8] Mavro, P., 2014. MariaDB High Performance. Packt Publishing Ltd.
- [9] Gackenhimer, C., 2015. Introduction to React. Apress.
- [10] Ohri, S., 2017. TypeScript 2.x By Example. Packt Publishing Ltd.
- [11] DuneBook. <https://www.dunebook.com/typescript-vs-javascript-why-typescript-is-next-to-big-thing/> (25.04.18)
- [12] Resende, D., 2015. Node.js high performance. Packt Publishing Ltd.
- [13] Kenler, E. and Razzoli, F., 2015. MariaDB Essentials. Packt Publishing Ltd.

Appendix

I. Source code

As this work is a project of Stagnation Laboratory, source code is not publicly available. To receive the source code of the developed application, please contact the author via the following e-mail: maria@stagnationlab.com.

II. License

Non-exclusive licence to reproduce thesis and make thesis public

I, Maria Belinska,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

Web application for managing Dag University, supervised by Helle Hein and Taavi Sangel,

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **14.05.2018**