

INTERNET TECHNOLOGIES – CW2

CONFERENCE APP

Linus Herterich - B00400078

Pawel Karpinski - B00360376

TABLE OF CONTENTS

Introduction	4
Overview	4
Background	4
Data Sources.....	4
Core Functions	5
Advertise	5
Register / Login	5
Ticket purchase	5
Booking workshops	6
User Dashboard.....	6
Advanced Functions	7
Register / Login Using Social Media	7
Admin Dashboard.....	7
Access to live or recorded events	7
Translation and content updates	7
Data Protection	8
Implementation.....	10
User Interface.....	10
Target group and design language.....	10
Mobile presentation.....	10
The homepage.....	11
Workshops	12
Talks.....	14
Tickets	14
Checkout Pages	15
Registration and Login	16
Profile page	17
Wireframe creation	18
Frontend implementation.....	18
Technology Stack.....	19
JavaScript.....	19
MERN Stack	20
Node.JS.....	21
MONGO DB	23
Express.js.....	23

REACT.js.....	26
Data Organisation	29
Hosting	31
DEPLOYMENT	31
Scalability	32
Tracking and Statistics.....	33
Conclusions	33
Bibliography	34

INTRODUCTION

Cinemas are in a crisis. Cinematography is booming nevertheless. This is because films are independent of crises, such as the current Corona crisis. More and more streaming providers are on the rise and services like "Netflix", "Amazon Prime Video" or "Disney+" are very successful. The cinemas may be empty, but the living rooms are full and people are eager for new films and series.

In order for new films and series to be created in times like these, filmmakers need and want to exchange ideas in order to stay networked as much as possible, to learn new things and to learn from colleagues or companies.

In the past trimester, we have been working on what a platform could look like so that film enthusiasts can also network and meet online within the context of a conference on cinematography. This report explains which functions we thought about in detail and how we implemented them in terms of design and technology.

OVERVIEW

BACKGROUND

CineCon 2021 is an International Cinematography online conference aimed for professional cinematographers. It is great opportunity to meet and connect with all talented people in the industry. Great talks and multiple workshops will expand arsenal of skills and broaden the creative horizons to tackle approaching uncertain and challenging times.

Carefully selected list of speakers will guarantee quality and invaluable insights about the industry, craft, and everchanging trends and technology.

DATA SOURCES

The data we mainly work with comes from the conference operators, the tutors of the workshops / talk speakers, as well as input from the conference visitors. We store all of this data in our private database, which can be filled by administrators using an API interface and can also be read by our frontend.

Furthermore, we work with licensed API interfaces from third parties, which handle core functionalities. For example, we have outsourced the entire payment logic to "Stripe" because it is both easier and safer to leave such things to professional specialised companies. In addition, the video streaming of talks and the operation of conference rooms is outsourced to professional suppliers. There are costs for all these APIs, but we would recoup them with the ticket prices.

CORE FUNCTIONS

ADVERTISE

As the webapp is basically public, each page should be designed to appeal to the target audience and promote the conference. Our content should be presented in such a way that it promotes new and existing conference visitors.

REGISTER / LOGIN

Interested users should be able to create an account on the webapp by registering via email and a password of their choice. Before doing so, they must accept the terms and conditions and privacy policy. A confirmation link will then be sent to them automatically by e-mail, which they must click on in order for the user account to be activated and ready for use.

Once the account is activated, a Log In is possible, which activates some functions on the website. Only after the login it is possible to buy tickets, book workshops and mark events as favourites. A personal user page can also only be accessed after logging in.

In addition to the user role, there is also an admin role, which allows administrators to create workshops / talks in a backend dashboard and to manage users and ticket purchases. The admin panel has not been implemented in the proof of concept yet. It is only possible to set up the role in the database administration, which enables a visit to the `"/admin"` page.

TICKET PURCHASE

We have decided to use a three-tier ticket system. The tickets differ in the number of workshops a conference participant can attend. The smallest and cheapest option contains 2 inclusive workshops, whereas the middle one contains four and the most expensive eight. All talks are unlimited inclusive for the user once any ticket has been purchased.

We have made this choice because cinematography is a very practical conference topic and there is more to be gained from workshops than from simple talks. However, you can only benefit from workshops if the number of participants is limited. This is also important for online conferences. Therefore, the limited resource are workshops, which is why we have integrated them into the ticket system.

Tickets can be booked directly on the website. The first requirement is a user account. As soon as this exists and the user clicks on "Book Now" on the ticket page, he is forwarded to the payment provider Stripe. Our backend creates a one-time checkout session for this, which contains the desired ticket option.

The payment processing itself is handled by Stripe. To test it out, the following test credit cards can be used for testing purposes:

NUMBER	BRAND	CVC	DATE
4242424242424242	Visa	Any 3 digits	Any future date
4000056655665556	Visa (debit)	Any 3 digits	Any future date
5555555555554444	Mastercard	Any 3 digits	Any future date
2223003122003222	Mastercard (2-series)	Any 3 digits	Any future date
5200828282828210	Mastercard (debit)	Any 3 digits	Any future date
5105105105105100	Mastercard (prepaid)	Any 3 digits	Any future date

(source: <https://stripe.com/docs/testing#cards>)

After the payment has been processed, Stripe forwards the user either to a page that confirms the success or failure of the payment. The ticket is then stored in the user's account and the user can now enrol in a set number of workshops depending on the ticket.

BOOKING WORKSHOPS

The workshops are stored in the database and can be created and edited flexibly and frontend-independently. Currently, this is only possible via the MongoDB backend. An admin interface for this is still missing.

The workshops can be booked in the frontend by logged-in users (if there are still bookings left from the ticket) or initially marked as favourites with a heart. The booked and favourite workshops can then be found on the profile page.

Unfortunately, we have not quite finished the booking system, which is why it is currently only possible to display the workshops from the database in the frontend, but not to book them with an account. The display on the profile page is currently only static and has nothing to do with real bookings or favourite markings.

USER DASHBOARD

Each user has their own dashboard where profile settings (unfortunately not implemented) as well as workshop bookings and favourite events can be viewed. If an event is about to start, it is highlighted and currently relevant information, such as the link to join the online conference, is displayed.

ADVANCED FUNCTIONS

The following features were all conceptualised during planning, but unfortunately did not make it into proof of concept.

REGISTER / LOGIN USING SOCIAL MEDIA

The login does not only work via registered email, but can also be done via platforms such as Google, Facebook or Apple. This would lower the entry barrier for many users to create an account on the website.

ADMIN DASHBOARD

A central interface where the entire conference can be managed. Workshops, talks and their speakers or tutors as well as users, their payments and bookings can be viewed and managed there. Furthermore, the livestreams for the talks can be started centrally and forwarded to the interested users by e-mail, and the workshops can be connected to conference platforms such as Microsoft Teams.

More detailed statistics, such as the workload of the website and the number of clicks on the talks and workshop visits, can also be viewed.

There is also a central place where notifications can be sent (either by push notification via the browser, or by email).

Admins can also block users (by IP address or by login name only), reset their passwords or nominate them as admins.

ACCESS TO LIVE OR RECORDED EVENTS

All talks have a direct connection to a streaming platform such as YouTube. However, the talks are not publicly accessible on this platform, but only by users of the conference. The talks can be accessed directly from the CineCon WebApp. Materials used in the talk (presentation slides or scripts) are also available on the WebApp.

If a user has missed a livestream, it can be viewed directly on demand on the website.

Workshops are only available live, but they are also accessible via the WebApp itself. As soon as a workshop is about to start, the link to join a conference system (Zoom, Hangouts, MS Teams, ...) is available on the WebApp and is sent to all participants by e-mail. This is done automatically, without the intervention of an administrator.

TRANSLATION AND CONTENT UPDATES

All texts and contents of the website can be translated and changed directly in the admin panel. Everything can be edited without changing the code. This means that the WebApp is not only suitable for CineCon 2021, but can also be used for any other conference.

This is already the case for the workshops. The basic workshop system is not bound to cinematography, as the texts and images can be replaced with anything.

DATA PROTECTION

Authentication of the user in our conference app is managed in the backend by `loginUser` function which checks email and password from request body and compare it with data stored in the db. Entered password is compared with its hashed version stored in the db using `bcrypt.compare()` method.

```
// checking if password is correct
userSchema.methods.matchPassword = async function (enteredPassword) {
  return await bcrypt.compare(enteredPassword, this.password);
};
```

If password match then new token is generated (valid for 7 days) and user's data is returned as a json response to the client.

```
180   if (user && (await user.matchPassword(password))) {
181     // generate token for the client valid for 7 days
182     const token = jwt.sign({ _id: user._id }, process.env.JWT_SECRET, {
183       expiresIn: "7d",
184     });
185     // collecting user data from db
186     const { _id, name, email, role, ticket } = user;
187
188     return res.json({
189       token,
190       user: { _id, name, email, role, ticket },
191     });
192   } else {
193     res.status(401).json({
194       error: "Email or password is wrong",
195     });
196     throw new Error("Invalid email or password");
197   }
198 };
```

On the client response is handled by `authenticate` helper method, token is stored in the cookie and user data is stored in the `localStorage`.

```
39 // authenticate the user by passing data to cookie and local storage
40 // during login
41 const authenticate = (response, next) => {
42   setCookie('token', response.data.token)
43   setLocalStorage('user', response.data.user)
44   next()
45 }
```



```

.then((response) => {
  // save the response (user and token) local storage/cookie
  authenticate(response, () => {
    // emptying the state
    setFormData({
      ...formData,
      email: "",
      password: "",
    });
  });
});

```

Authorisation is a level of privileges according to type of the user. In our project we have 2 types of users: 'user' and 'admin' And depending of type one is able to access different parts of the application. Component AdminRoute was created to check and allow access only for 'admin' members

```

const AdminRoute = ({ component: Component, ...rest }) => (
  <Route
    {...rest}
    render={(props) =>
      isAuth() && isAuth().role === "admin" ? (
        <Component {...props} />
      ) : (
        <Redirect
          to={{
            pathname: "/login",
            state: { from: props.location },
          }}
        />
      )
    }
  ></Route>
);

export default AdminRoute;

```

Other security measures are hashing and salting passwords before storing them in the database. So even admins don't have access to user's passwords.

```

// crypt password before save
userSchema.pre("save", async function (next) {
  if (!this.isModified("password")) {
    next();
  }
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
});

```

```

password: "$2a$10$J8cnjco.Ni3ob9ix3mXj6.z2gL3hm20BwVKChw3YrgKowxc3qwxE2"
createdAt: 2021-01-19T23:00:45.532+00:00

```

IMPLEMENTATION

USER INTERFACE

TARGET GROUP AND DESIGN LANGUAGE

Before we started designing the interface, we thought about the target group we wanted to address. Since we want to promote a conference about cinematography, the target group of the website consists of people interested in cinema and moving images of any kind.

The aim is to reach beginners who want to enter the field with the help of the conference, as well as professional business people and industry insiders. This balancing act between information for beginners and information for professionals is reflected in both the content and the interface design of the website.

Basically, we decided on a modern and consistent design that picks up on elements of material design. As a primary colour we have chosen a turquoise (#007c7e). As a cool colour, this exudes professionalism but also refreshment. As a conference, we want to present ourselves professionally, as we want to address professionals in the industry and also convey professional information in the conference. Nevertheless, we don't want it to look too boring, because the industry generally works with imagery and has a good visual look. Thus, the green has a good influence on the website being perceived as refreshing and new.

For the font, we decided on a sans-serif typeface that radiates clarity and cleanliness as well as professionalism. We work with the primary colour, kerning and font weight for certain typefaces.



CINECON 2021 TICKETS
GET YOUR ACCESS

The image shows a typographic design for 'CINECON 2021 TICKETS'. The main headline 'GET YOUR ACCESS' is displayed in a large, clean, sans-serif font. 'GET YOUR' is in black, while 'ACCESS' is in the primary turquoise color. Above it, 'CINECON 2021 TICKETS' is written in a smaller, all-caps, spaced-out sans-serif font.

Figure 1: Typefaces Design

So that it does not become too text-heavy, we have always tried to insert pictures or illustrations in appropriate places. It was also important to us to always use enough white space so that the page does not appear too cluttered and confusing. Thematically related elements were either separated as a section with their own background colour (usually nuances of the primary colour) or with card elements from the material design.

MOBILE PRESENTATION

As this is an online conference that is best consumed via laptop, tablet or desktop PC, the presentation of the web app was also designed for these device types first. Nevertheless, a mobile version is possible without further difficulties. All elements can also be displayed very well on smaller displays. However, due to time constraints, an optimised design specifically for smartphone displays was not created.

THE HOMEPAGE

The homepage welcomes with a large hero image in the primary colour of the website showing a film reel. The headline simply shows what it is about and directly presents the date of the conference. The call-to-action button immediately leads to the ticket purchase.

Right after that, we want to create a "WOW" effect by listing the logos of the companies participating in the conference. Since these are leading industry giants, we are positioning ourselves to the visitor right away as a website that has a name in the industry. So, we are not a niche conference, we are relevant. This should be clear to the user here.



Figure 2: Homepage Hero Element



Figure 4: Listing participating companies

FEATURED GUEST SPEAKERS

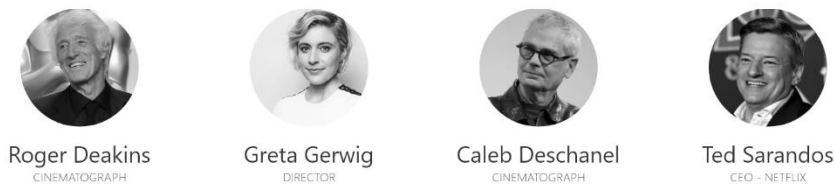


Figure 3: Highlighting guest speakers

Immediately afterwards, this effect is repeated by highlighting 4 guest speakers of the conference who have a name in the cinematography world. Not only CEOs nor only directors were picked deliberately, so that one can see a wide range of offers directly on the homepage.

Then the visitor is addressed directly for the first time and presented with 3 reasons why he should attend the conference. The visitor's gaze is first caught here by large icons. If he wants to read more, he reads the reason and if more reasons are needed, he can be redirected to the "About" page via the "more reasons" button.

All this was initially to attract new visitors to the conference. Now follows an excerpt of this year's event plan, which should convince

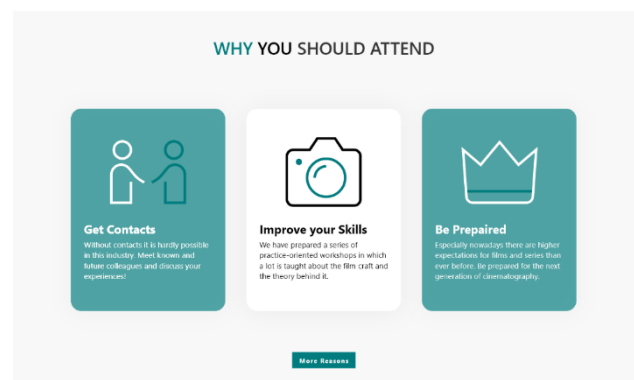


Figure 5: Reasons for attending the conference

returning visitors to come to the 2021 conference. The two buttons "See all Talks/Workshops" indirectly indicate that this is not the entire event plan, but only an excerpt. If someone wants to find out more, he or she can click on one of the buttons here and land on the pages that provide the entire event plan.

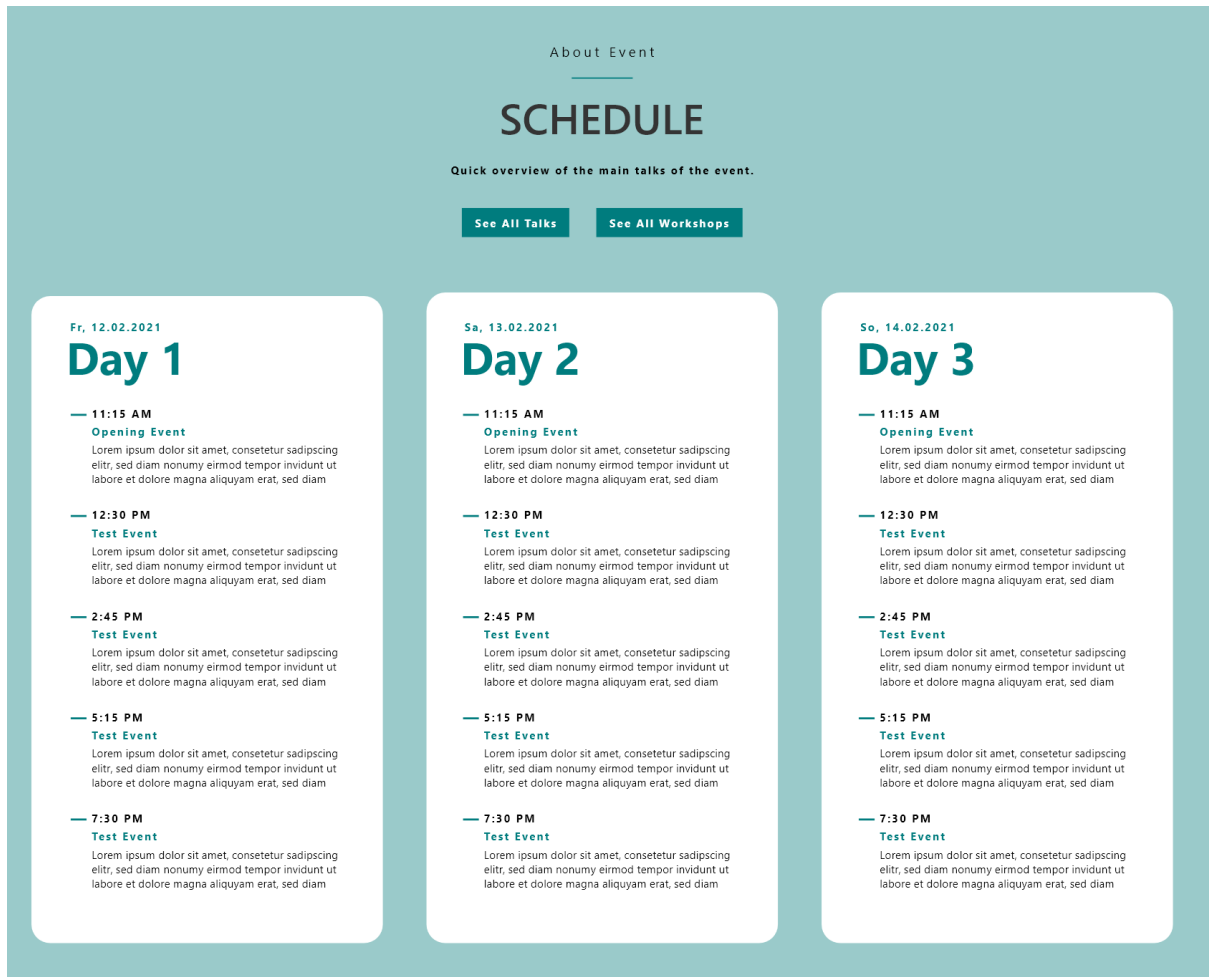


Figure 6: Event Schedule

WORKSHOPS

On this page, all workshops are presented in a visually appealing way in Material Design Cards. All events are categorised into topics. These can be found on the bottom line of the workshop cards. Buttons next to the sorting option can be used to select specific subject categories, which are then presented at the bottom. Thus, a selection is possible for the user. This is quite important, as the field of cinematography is very diverse and also contains many different professions, all of which are distinct. We offer workshops for different professions here, so that the conference is also suitable to be attended by a wide range of people. To the right of the category filters there is also a

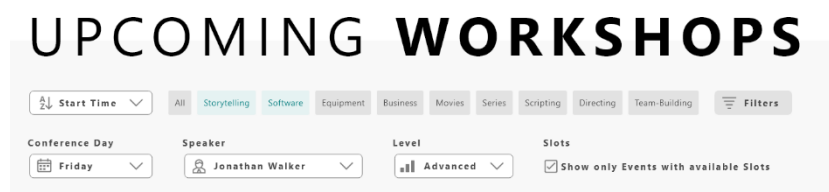


Figure 7: Filter Options for Workshop Page

button which activates further filters. These are initially hidden. If they are shown, the events can be filtered here only for certain days or by certain tutors.



Figure 8: Workshop Card

The workshop cards themselves contain a small picture to attract attention, a title, a short description and also information about the tutor, the starting time and the level of difficulty. Here again we pick up beginners as well as professionals, as it is clear that there are both beginner events and advanced events. The events can be marked as favourites or opened using the two buttons. The focus button here is on the button that opens the event to find out more information and to book the event there. The workshop cards also show how many slots are still available. This should show the shortage of workshops and encourage the viewer to buy his ticket as soon as possible, because otherwise his favourite workshop might already be fully booked.

The detail page of the workshops works as a modal (pop-up) and contains the workshop information in more detail. If the user is logged in, the event can be booked directly there if there are still workshop bookings left from his ticket.

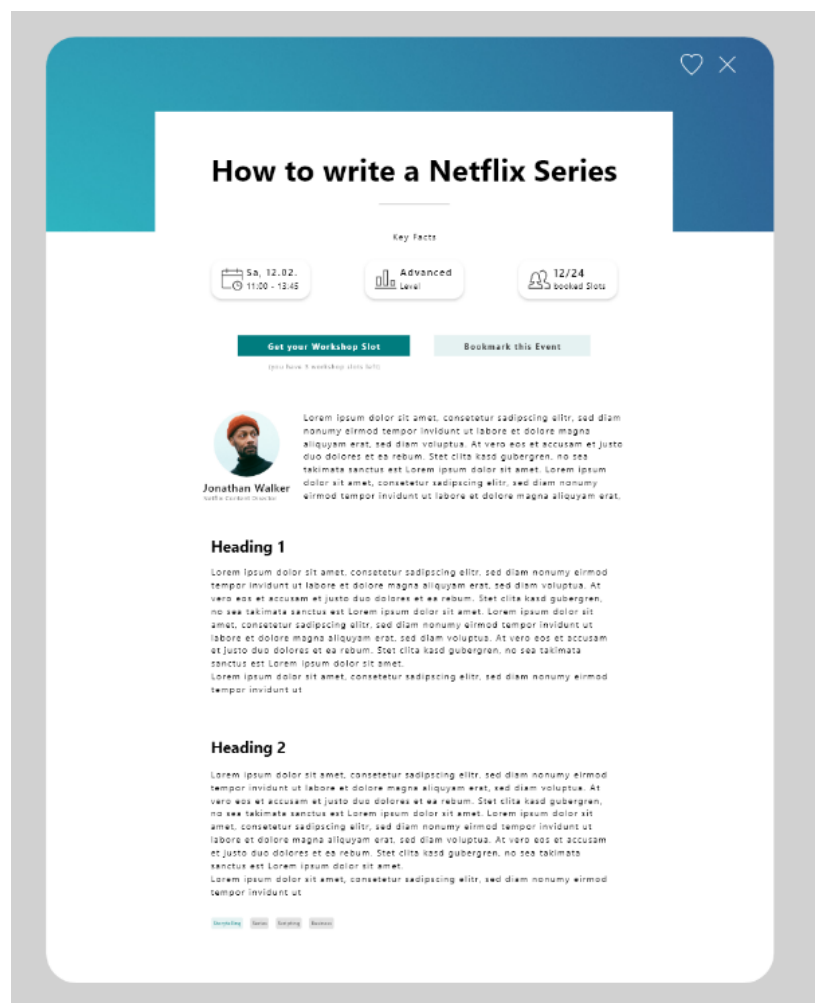


Figure 9: Workshop Details (Modal)

TALKS

Similar to the workshops, the talks are also designed with material design cards on which the short description and the categories can be found. As the focus of the conference is more on the workshops, there are not so many Talks, which makes filtering not as relevant as with the workshops. Due to a lack of time, we did not continue working on the design for the Talks page after the first conception. The page was then also not included in the app.

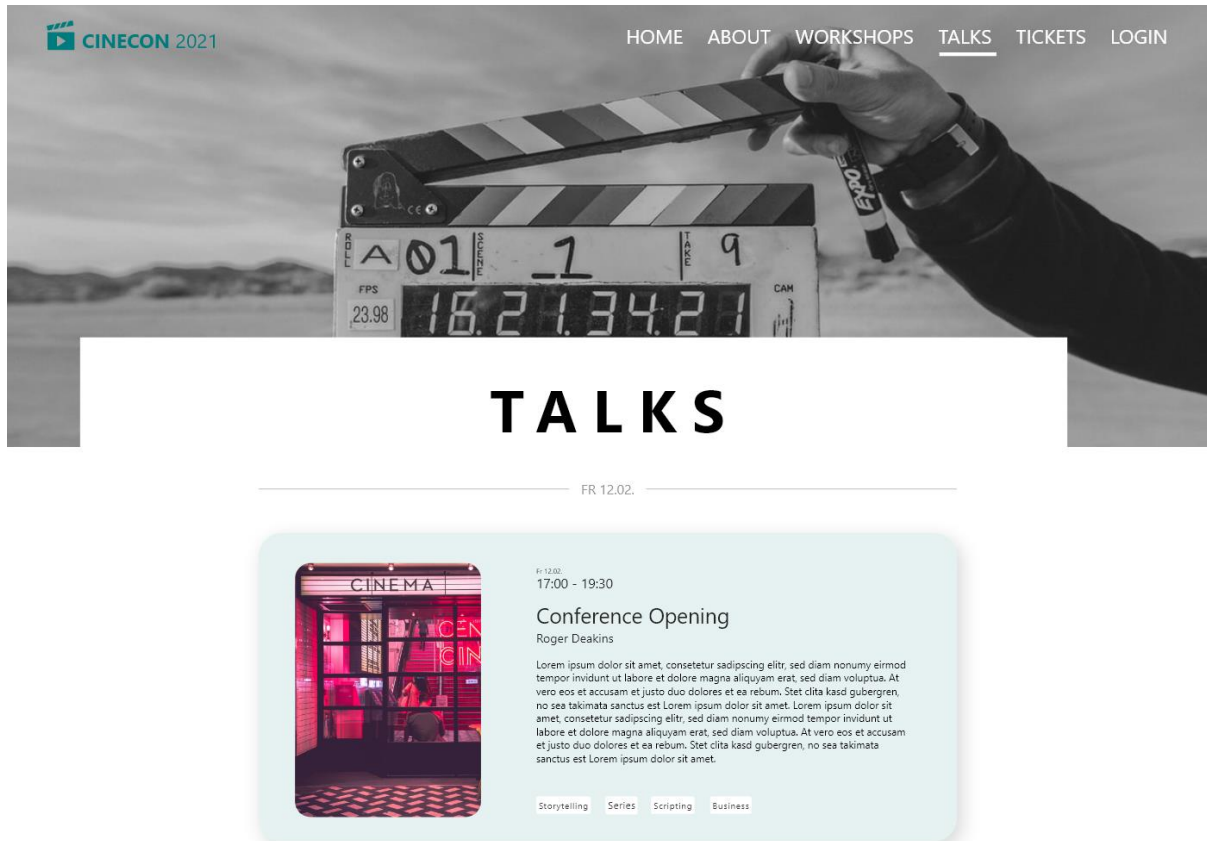


Figure 10: Talks page - concept

TICKETS

The ticket page focuses primarily on highlighting the differences between the individual tickets. Thus, the three ticket options (Small, Medium, Large) are lined up in an almost identical style, making it as easy as possible to compare the tickets. However, there is a visual focus on the Medium ticket. This has the most saturated colour and thus attracts attention. Psychologically, the golden mean is most likely to be chosen if there is still a slim option that is not much cheaper but holds many fewer options. According to the motto "I can afford the small surcharge". By contrast, the expensive ticket would be beyond the scope of the

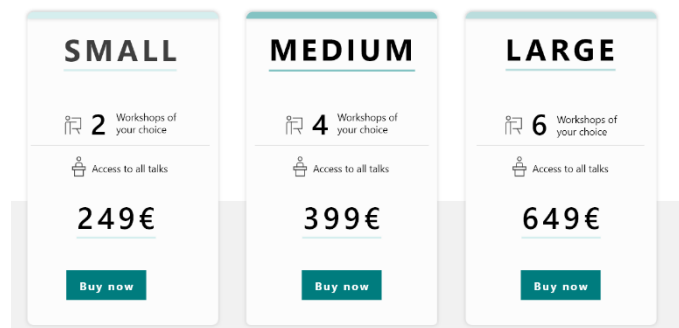


Figure 11: Ticket comparison

normal user and the thought spreads "I don't need that, it's too expensive and over the top". Nevertheless, this option is probably booked by enthusiasts who want to use the conference on all trains.

At the bottom of the page is a FAQ section, which also follows the card design of the entire website. Questions that prevent the user from buying a ticket should be solved directly so that nothing stands in the way of the purchase.

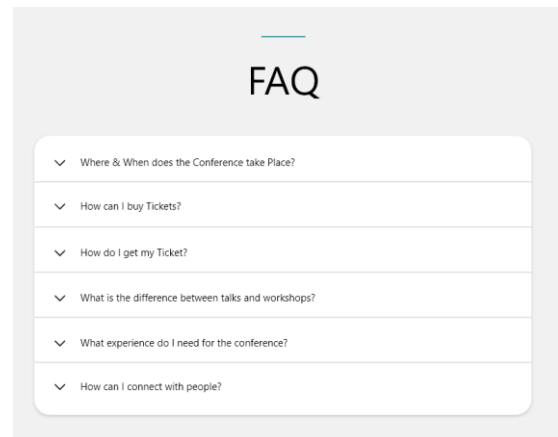


Figure 12: FAQ Section

CHECKOUT PAGES

After the transaction with our payment provider "Stripe" has been either successful or unsuccessful, the user lands on one of the two pages:

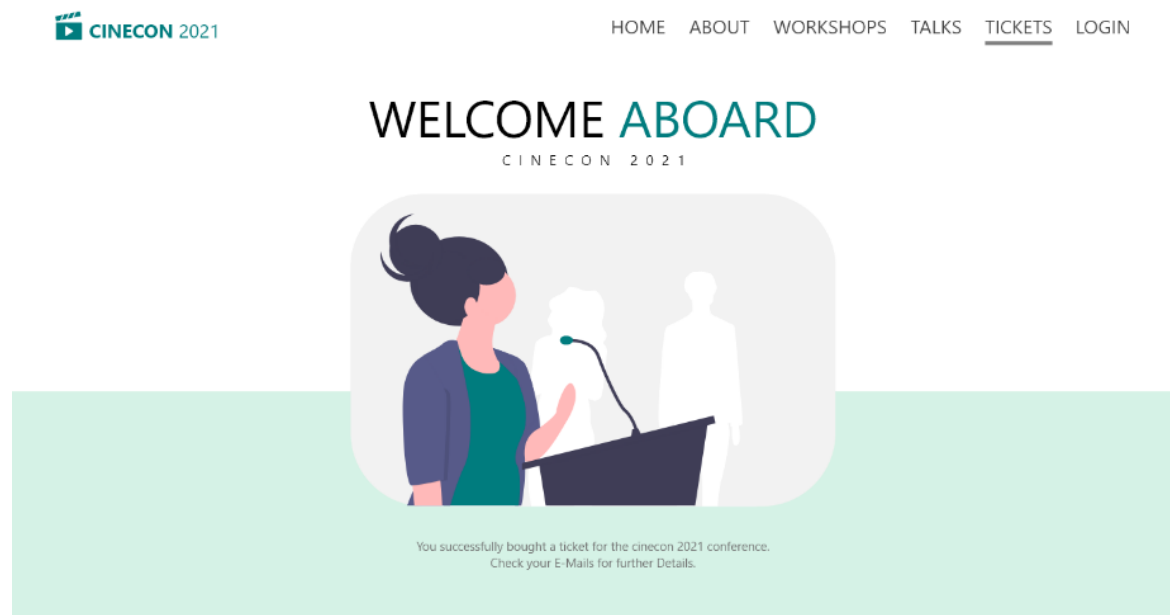


Figure 13: Successful Checkout

Here we congratulate the user, reinforcing the feeling that they have made the right choice. He is then asked to check his inbox to find out what happens next. It is important at this point that the user is not lost as to what to do next, but is given information on what to do now.

Something went wrong...

CINECON 2021

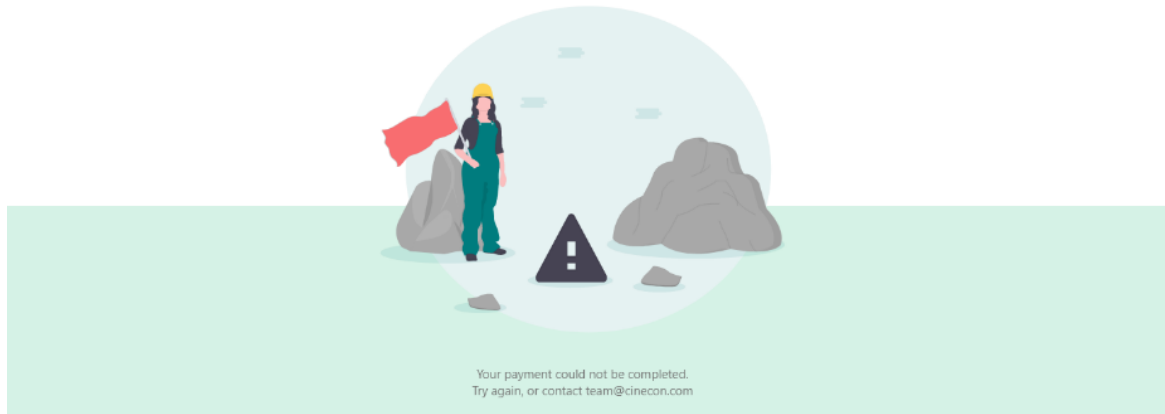



Figure 14: Failed Checkout

If something goes wrong during the checkout process at Stripe, the user is taken to a page that also tells them what to do next. Either try again or report it to a contact person. This means that the customer is not just told that something has gone wrong, but is immediately presented with a possible solution.

REGISTRATION AND LOGIN

CINECON 2021

Join our worldwide conference for professional cinematographers! Meet talented colleagues and enjoy impressive speeches together.



Sign Up

FIRST NAME **LAST NAME**

E-MAIL ADDRESS

PASSWORD

Yes, I want to receive Cinecon marketing and update e-Mails

Yes, I agree all the [Terms](#) and the [Privacy Policy](#)

Already have an Account? [Log In](#)

Figure 15: Registration Page

The registration and login page has been visually combined and can be switched by clicking a link under the "Sign Up/In" button. The registration does not require a lot of information that could otherwise discourage an interested visitor from registering. The page has been designed as clearly as possible and the individual fields have been described with both labels and placeholders.

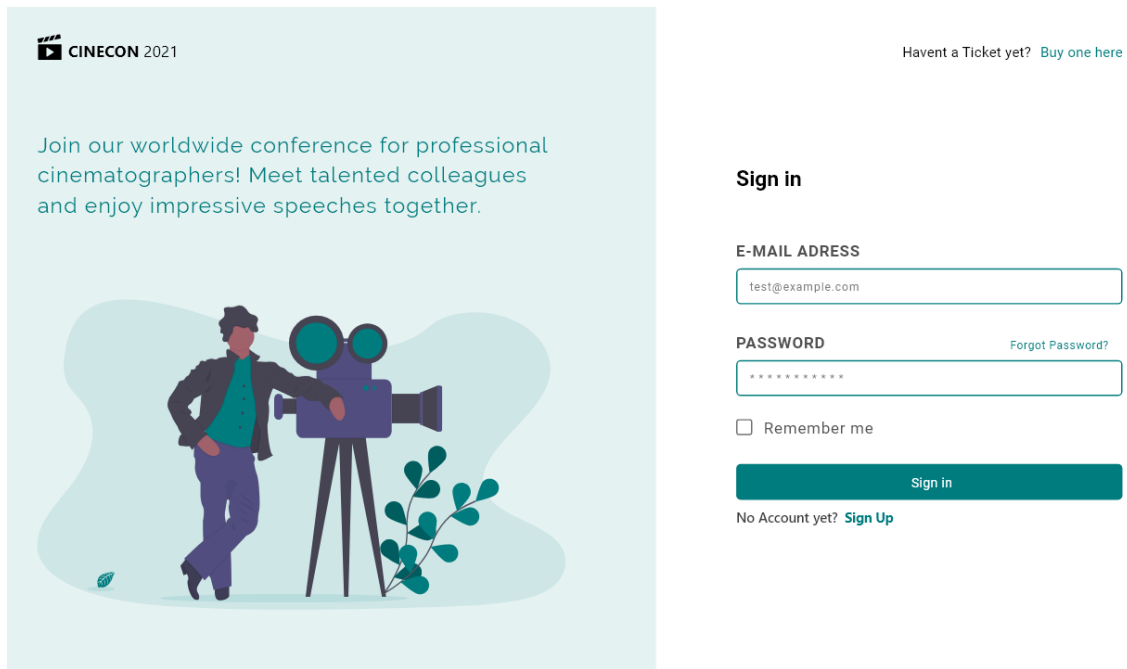


Figure 16: Sign In Page

PROFILE PAGE

Here the user is first greeted personally by their first name, which emphasises that this is their own profile page and not a public page. If the user already has a ticket, details about it will be posted here.



Figure 17: Profile - Welcome Element

Further down, the events booked by the user are listed in a more abbreviated form than on the workshop page. Here it is more about an overview of the upcoming dates than about the details of the workshops. Nevertheless, the details can be called up via a button, or the event can be cancelled. On the right side you will find the favourite events in an even smaller format.

Your booked Events



Your favorite Events

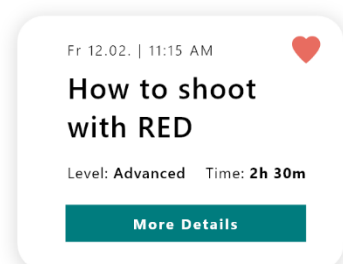
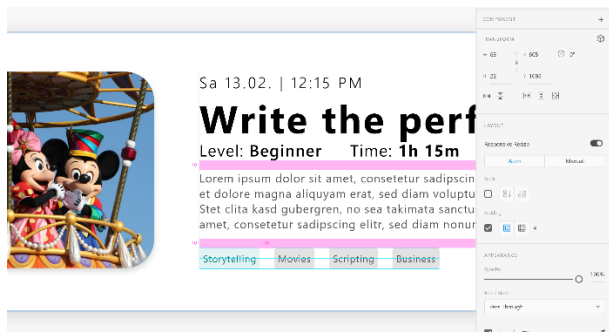


Figure 18: Booked Events & Favourite Events

WIREFRAME CREATION



After we were clear about the features and had clarified the basic style of the webapp, we created the wireframes in Adobe XD. For this purpose, we used a shared document, which can also be viewed online here:

<https://xd.adobe.com/view/b9414fb9-a536-4020-86bb-320ed6892acb-ec12/grid?grid>

Figure 19: Wireframing in Adobe XD

FRONTEND IMPLEMENTATION

We implemented the frontend with the help of the CSS library Bootstrap (version 4.5). Since we chose React as our frontend framework, we implemented Bootstrap with the help of the "React-Bootstrap" rebuild. This transforms the most common Bootstrap components into native HTML elements.

For example, the following native HTML-Bootstrap Code:

```
<div class="card">
  <div class="card-body">
    This is some text within a card body.
  </div>
</div>
```

Will be in Bootstrap-React:

```
<Card>
  <Card.Body>
    This is some text within a card body.
  </Card.Body >
</Card>
```

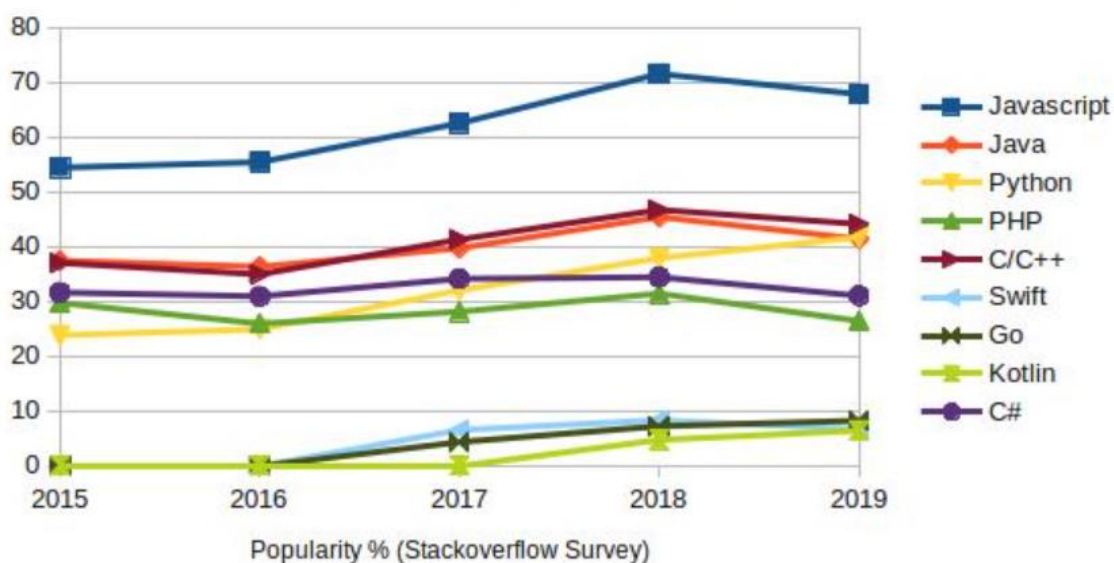
This makes the entire code much easier to read for the components (or JSX) and integrates well into the usual React syntax.

TECHNOLOGY STACK

JAVASCRIPT

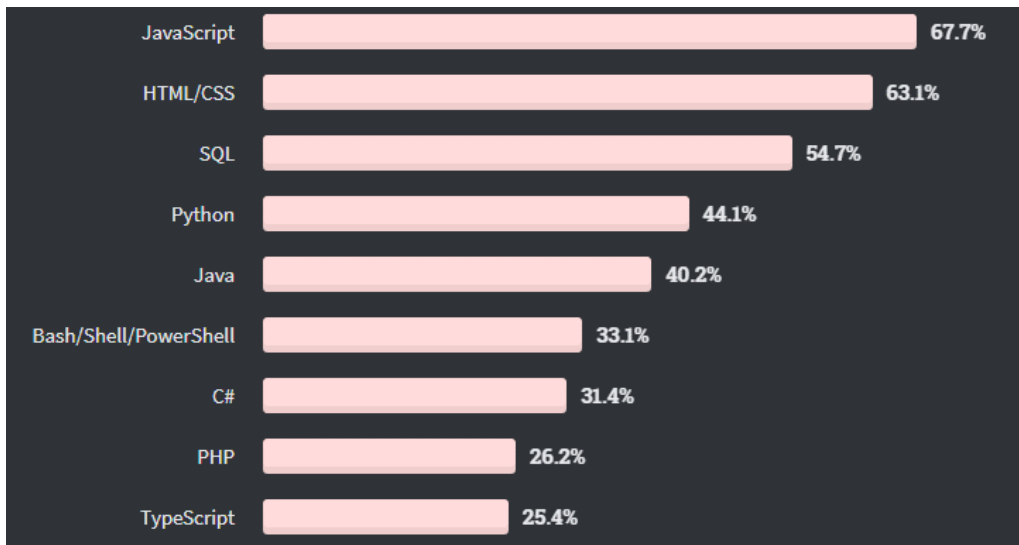
For the application technology stack, we decided to focus on one programming language – JavaScript. Debuted in 1995 with Netscape Navigator 2 web browser, created to add some life into mostly static and dull pages. After years of development and improvements mostly introduced in ECMAScript6 (ES6) update JavaScript became fully grown language able to create sophisticated applications. From beautiful, interactive, and dynamic websites, mobile or desktop applications to Internet of Things (IoT) devices.

JavaScript takes the first spot on the popularity charts several years in the row. It is popular choice because it can be utilised on both 'sides' of the application – client side (frontend) for creating user interfaces and server side (backend) thanks to Node.js (runtime environment for JS outside the browser). There are many open-source, free libraries and frameworks which can shorten the development time and therefore save money for the companies. There are whole stacks build entirely on JS technologies used for development, and the most popular are: MERN, MEAN, MEVN. As well front-end frameworks and libraries like React, Angular, Vue which can work wit any backend technology (.NET, Java, PHP, Ruby).



Programming Language Popularity 2015-2019

Source: <https://codinginfinite.com/top-programming-languages-2020-stats-surveys/>

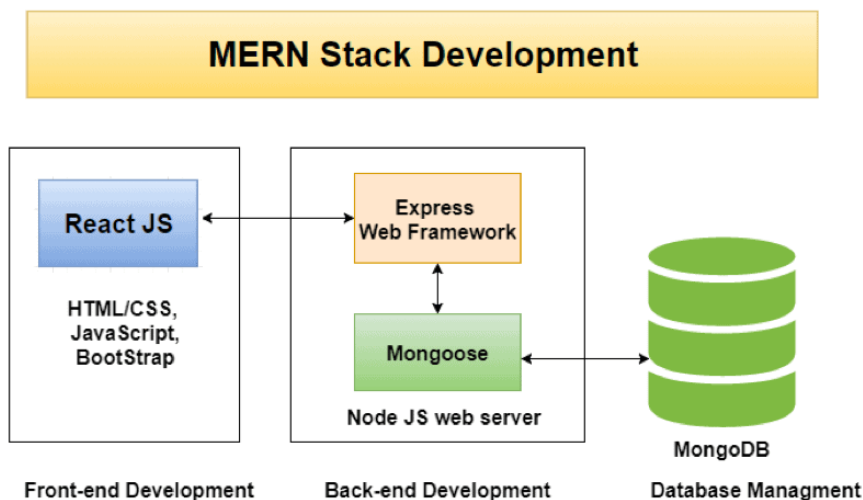


Programming, Scripting, markup languages popularity 2020 (stackoverflow survey)

Source: <https://insights.stackoverflow.com/survey/2020#most-popular-technologies>

MERN STACK

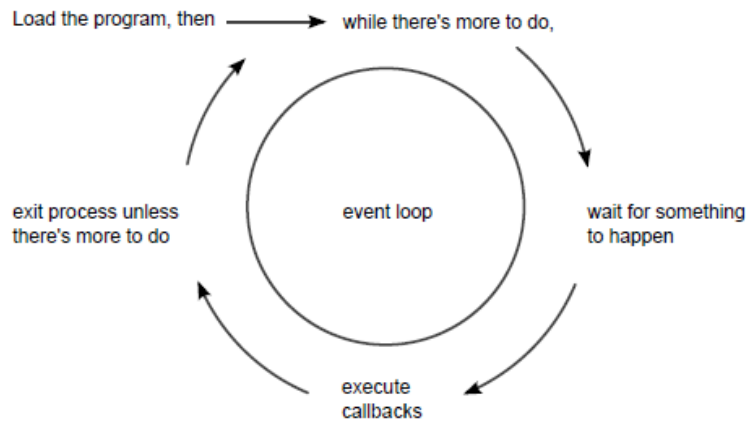
As above charts show JavaScript is still relevant and popular choice for developers in 2020, therefore we tried to implement **MERN** stack – **MongoDB** for database, **Express.js** for HTTP server, **React.js** library for user-interface and **node.js** as runtime and development environment. It is great example of “JavaScript everywhere” paradigm - Combining web application development around one programming language. (J., 2013)



Source: <https://www.bocasay.com/how-does-the-mern-stack-work/>

NODE.JS

Node.js allows to execute JavaScript code outside the web browser, it is a runtime open-source, cross-platform environment built on Google's V8 engine. Created by Ryan Dahl in 2009 and still in development (latest version is 15.3 – 24.11.20) let developers create command line tools and server-side scripts using JavaScript. Node's reacts and consume events (event – driven architecture), processes asynchronous input / output and can run non blocking code – able to run next tasks without finishing previous (fetching, data, API requests). 'Node's multitasking is achieved by event-loop, it processes events from the queue by running callbacks on those events. (Subramanian, 2019)



Node Event loop

Source: (Wilson, 2018)

There are several node packages used in our application they are stored in node_modules folder. List is available in **package.json** file that contain all important information about the project, node scripts which can start backend or client or both servers and script used for the deployment of the application to Heroku hosting. List of dependencies (node packages): used:

```
"dependencies": {
  "@sendgrid/mail": "^7.4.2",
  "bcryptjs": "^2.4.3",
  "body-parser": "^1.19.0",
  "colors": "^1.4.0",
  "cors": "^2.8.5",
  "dotenv": "^8.2.0",
  "express": "^4.17.1",
  "express-async-handler": "^1.1.4",
  "express-jwt": "^6.0.0",
  "express-validator": "^6.9.2",
  "jsonwebtoken": "^8.5.1",
  "mongoose": "^5.11.8",
  "morgan": "^1.10.0",
  "path": "^0.12.7",
  "stripe": "^8.131.1"
},
"devDependencies": {
  "concurrently": "^5.3.0",
  "nodemon": "^2.0.6"
}
```

Package Name dependencies	Function
@sengrid/mail	Mail Service for the SendGrid v3 Web Api. Sendgrid service is used to send activation link for the registered user. Link is active only for 10minutes. It is use as a precaution for mass registering of fake accounts.
Bcryptjs	Used for hashing user passwords stored in the database.
Body-parser	Parse incoming request bodies before handlers
Colors	Add colours to command line messages for better development experience
Cors	Middleware that enables CORS (Cross-origin resource sharing). CORS is a mechanism that allows restricted resources on a web page to be requested from another domain (outside the original domain)
Dotenv	Used for loading environmental variables (usually sensitive data are kept there) stored in .env file
Express	Web application framework
Express-async-handler	Handling errors inside of express async routes
Express-jwt	Express middleware for validating JSON Web Tokens through jsonwebtoken module
Jsonwebtoken	JSON Web tokens handling signing and verification
Express-validator	Express middleware for validator.js library which validates and sanitizes strings
Mongoose	MongoDB object modelling tool designed to work in asynchronous environment
Morgan	HTTP request logger middleware useful during development
Path	Used for handling and transforming file paths
Stripe	Node library for Stripe API (secure payments)
Concurrently	Dev dependency used to run multiple commands at the same time. Used in development to start server and client with one command
Nodemon	Used in development restarts node file when detecting any changes that has been made.

MONGO DB

MongoDB is a NoSQL database with flexible schema and JSON-based query language. It often called document database. Opposing to traditional relational database where we have tables with rows and columns MongoDB has collections of documents which are just json objects.

We're using MongoDB Atlas global cloud database hosted by Amazon Web Services which guarantee availability, scalability and compliance with very demanding security and privacy standards. DB Compass is used as a graphical user interface for MongoDB and it let view, edit and analyse collections without knowing of query syntax, besides it can be also used to optimise query performance, manage indexes, and document validation.

EXPRESS.JS

According to developers Express.js is a “fast, unopinionated, minimalist, web framework for Node.js”. (OpenJS Foundation, 2020) It vastly reduce time and code needed to create HTTP server. It handles requests and routes and utilises middleware functionality (enabling components to work together).

In our conference app backend code is contained in **server.js**, app is an express() package and app.use incorporates any middleware we want to use with express (imported routes for example)

```
// importing routes
import authRoutes from "./routes/authRoutes.js";
import userRoutes from "./routes/userRoutes.js";
import workshopsRoutes from "./routes/workshopsRoutes.js";
```

```
const app = express();
app.use(express.json());

// middleware
app.use("/api", authRoutes);
app.use("/api", userRoutes);
app.use("/api", workshopsRoutes);
```

authRoutes file contain all routes for HTTP request and middleware methods for validation and handle database queries.

```

// routes
// sign-up
router.post(
  "/signup",
  userSignUpValidator,
  runValidation,
  signupUserWithSendGrid
);

// log-in
router.post("/login", userLoginValidator, authUser);

// account email activation
router.post("/activate", accountActivation);

// payment
router.post("/checkout", checkoutPayment);

// Checkout Event Webhook
router.post("/webhook", checkoutWebhook);

// Checkout Event Webhook
router.post("/activateTicket", checkoutActivate);

```

authController file contain methods used by router as a middleware. There is example of signing up new user using SendGrid, a Twilio service for email delivery. This is used for sending activation link for the newly registered users, if user won't respond in 15 minutes link will expire and account wont be stored in the database. This is use to prevent registering fake users and large scale signing by bots.

User.findOne({email}) is a mongoose method to find data by entered parameter. We checking for existing emails in the database is email is already registered then error message is send back to client if no then new token is created for email activation (**const token = jwt.sign()**). Then token is send as a part of the link for registered user's email. Message is displayed back in the client that there is 15 minutes to activate an account.


```

46 v const signupUserWithSendGrid = (req, res) => {
47   // getting data from request body
48   const { name, email, password } = req.body;
49
50   // checking database for existing email
51 v User.findOne({ email }).exec((err, user) => {
52 v   if (user) {
53 v     return res.status(400).json({
54       error: "Email is taken",
55     });
56   }
57 });
58
59   // creating token for email activation
60 v const token = jwt.sign(
61   { name, email, password },
62   process.env.JWT_ACCOUNT_ACTIVATION,
63   { expiresIn: "15m" }
64 );
65
66   // creating activating email
67 v const emailData = {
68   from: process.env.EMAIL_FROM,
69   to: email,
70   subject: "Account activation link",
71 v   html: `
72     <h1>Cinecon 2021 account activation</h1>
73     <p>Please use this link to activate your account</p>
74     <p>${process.env.CLIENT_URL}/auth/activate/${token}</p>
75     <p>${process.env.CLIENT_URL}</p>
76     `,
77   };
78
79   // sending an email
80   sendGridMail
81     .send(emailData)
82 v     .then((sent) => {
83 v       return res.json({
84         message: `Activation email has been sent to ${email}. Activation link will
85           expire in 15 minutes`,
86       });
87 v     }.catch((err) => {
88       console.log(err);
89     });
90   });

```

Controllers use Models which are responsible for creating and reading documents (instance of a model) from MongoDB database. Mongoose model is a wrapper on the mongoose Schema which defines the structure of the document, default values and validators. Model provides the interface to the database queries and CRUD operations.

```
const userSchema = new mongoose.Schema({
  name: {
    type: String,
    trim: true,
    required: true,
    max: 32,
  },
  email: {
    type: String,
    trim: true,
    required: true,
    unique: true,
    lowercase: true,
  },
  password: {
    type: String,
    required: true,
  },
  role: {
    type: String,
    default: "user",
  },
  ticket: {
    type: String,
    default: "",
  },
}, { timestamps: true });
```

REACT.JS

React is a JavaScript library for building user interfaces created by developers behind Facebook in 2013. Still in development (latest version 17 released in 10/20). 'React's declarative and modular nature makes it easy for developers to create and maintain reusable, interactive, and complex user interfaces.' (Hoque, 2020). React is a great choice when comes to modularisation and reusability of the UI components. JSX – JavaScript XML allows to create HTML elements and render it the DOM without need of createElement() and appendChild() methods. React code without JSX become hard to read when number of nested elements grow, therefore JSX is a 'syntactic sugar for the React.createElement(, component, props, ...children) function.' (reactjs.org, 2020)

React frontend application was created using CLI tool create-react-app – which setups nice React development environment without hassle of configuring build tools (webpack, babel, etc.) Some additional packages have been installed to extends app functionality:

```
"name": "frontend",
"version": "0.1.0",
"private": true,
"dependencies": {
  "@stripe/stripe-js": "^1.11.0",
  "@testing-library/jest-dom": "^5.11.8",
  "@testing-library/react": "^11.2.3",
  "@testing-library/user-event": "^12.6.0",
  "js-cookie": "^2.2.1",
  "react": "^17.0.1",
  "react-bootstrap": "^1.4.3",
  "react-dom": "^17.0.1",
  "react-router-bootstrap": "^0.25.0",
  "react-router-dom": "^5.2.0",
  "react-scripts": "4.0.1",
  "react-toastify": "^6.2.0",
  "stripe": "^8.131.1",
  "web-vitals": "^0.2.4"
},
```

Package Name dependencies	Function
@stripe/stripe-js	Wraps global Stripe function Stripe.js as an ES module, used for payment
Js-cookie	JavaScript Api for handling cookies used for store user's authentication token information in the browser's cookie
React-bootstrap	Bootstrap 4 components built with React
React-router-bootstrap	Integration between react Router and react-bootstrap
React-router-dom	DOM bindings for react-router
React-toastify	Used for notifications sent by server
Stripe	Provides access to Stripe API from applications written in server-side JavaScript
Axios	Promise based HTTP client for the browser

Our App consist of multiple components which are required by different views (pages) and then everything is gathered inside App functional component which uses BrowserRouter from react-router-dom to emulate behaviour of multiple pages even though this is a single page application. Components <Header /> and <Footer /> are shared across whole application and only content between them is changing depends of browser's url address.

```

19  const App = () => {
20    return (
21      <Router>
22        <Header />
23
24        <Route path="/" component={HomeScreen} exact />
25        <Route path="/workshops" component={WorkshopsScreen} exact />
26        <Route path="/about" component={AboutScreen} exact />
27        <Route path="/tickets" component={TicketScreen} exact />
28        <Route path="/checkout" component={CheckoutScreen} exact />
29        <Route path="/login" component={LoginScreenFunc} exact />
30        <Route
31          path="/auth/activate/:token"
32          component={ActivateAccountScreen}
33          exact
34        />
35        <ProfileRoute path="/profile" exact component={ProfileScreen} />
36        <AdminRoute path="/admin" exact component={AdminScreen} />
37        <Footer />
38      </Router>
39    );
40  };
41
42  export default App;

```

Example of single home page (view) which imports separate components to create whole layout.

```
1 import React from "react";
2 import HomeHero from "../components/HomeHero";
3 import Sponsors from "../components/Sponsors";
4 import Speakers from "../components/Speakers";
5 import Reasons from "../components/Reasons";
6 import Schedule from "../components/Schedule";
7
8 const HomeScreen = () => {
9   return (
10    <>
11      <HomeHero />
12      <Sponsors />
13      <Speakers />
14      <Reasons />
15      <Schedule />
16    </>
17  );
18 };
19
20 export default HomeScreen;
```

We use React hooks to use state of the application without need of writing a class component. Most popular hook used here are useState() - for storing authenticated user data and

```
const LoginForm = () => {
  // state with useState hook
  const [formData, setFormData] = useState({
    email: "",
    password: "",
  });
  // destructuring state
  const { email, password } = formData;
```

dynamic render of the components depending of state's information. Another popular hook is useEffect() which is triggered every time components is rendered or some data in the components change.

```
useEffect(() => {
  const fetchWorkshops = async () => {
    const result = await axios.get(`${process.env.REACT_APP_API}/workshops`);
    setWorkshops(result.data);
  };

  fetchWorkshops();
}, []);
```

In the above's example useEffect() is used to get data of the available workshop's from the MongoDB using axios. setWorkshops() method is used to store state (all fetched workshops info) which can be passed as a props to <Workshop /> component.

```

<Container>
  {workshops.map((workshop) => (
    <Workshop workshop={workshop} />
  ))}
</Container>

```

Map() higher order array method is used to loop through workshops state and for each found workshop render <Workshop /> component where rest of the data is passed as a workshop prop. Below is <Workshop /> component which renders all workshop information fetched from db like {workshop.date} or {workshop.title}.

```

<Col className={!props.small ? "ml-2 col-lg-7" : "ml-2 col-lg-7"}>
  <p>
    {workshop.date} | {workshop.time}
  </p>
  <h4 className="text-dark">{workshop.title}</h4>
  <div className="d-flex">
    <p className="">
      Level: <strong>{workshop.level}</strong>
    </p>
    <p className="ml-3">
      Duration: <strong>{workshop.duration}</strong>
    </p>
  </div>

```

DATA ORGANISATION

Plan for our conference app was to create 4 main collections where we can store information about registered users, available tickets, workshops, and talks.

In addition to the main data, we would have smaller data sets that would be linked to the main data by ID to avoid the redundancy of repeating strings. Categories (ID and Name) for the Workshops:

- Tutors (ID, Name, Profession, Description, Image) for the Workshops
- WorkshopLevel (ID, Name) for the Workshops
- Speakers (ID, Name, Profession, Description, Image) for the Talks
- User Roles (ID, Name) for Users e.g.: admin, user

Due to time restrictions final proof of concept is lacking some of the data and functionalities.

User document from users collection from MongoDB Compass:

```
_id: ObjectId("600b528fc1ed3911b08f6584")
role: "user"
ticket: ""
name: "Filip"
email: "filipk490@gmail.com"
password: "$2a$10$mhtBWT3boaEgW7q/1ZS84.t0Xr.SLCq9W71eu6aDzddFXw4Tyroty"
createdAt: 2021-01-22T22:32:47.363+00:00
updatedAt: 2021-01-22T22:32:47.363+00:00
__v: 0
```

Workshop document (workshops) collection from MongoDB Compass

```
_id: "1"
title: "How to write a NETFLIX series"
image: "/images/workshop_1.jpg"
description: Object
  short: "Lorem ipsum dolor sit amet consectetur adipisicing elit. Eligendi amet..."
  detail: Array
    0: Object
      heading: "Know the Basics about Netflix"
      text: "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed diam nonu..."
    1: Object
      heading: "Writing Style Optimization"
      text: "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed diam nonu..."
  tutor: Array
    0: Object
      name: "John Walker"
      job: "Netflix Content Director"
      image: "/images/tutor-1.png"
      description: "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed diam nonu..."
  slots: 30
  booked: 6
  category: Array
    0: "Storytelling"
    1: "Series"
    2: "Scripting"
    3: "Business"
  duration: "2h 30min"
  level: "Advanced"
  date: "Fr 12.02"
  time: "11:15AM"
```

Workshop model and Schema

```

1 import mongoose from "mongoose";
2
3 const workshopSchema = new mongoose.Schema(
4   {
5     title: {
6       type: "String",
7     },
8     image: {
9       type: "String",
10    },
11    description: {
12      short: {
13        type: "String",
14      },
15      detail: {
16        type: ["Mixed"],
17      },
18    },
19    tutor: {
20      type: ["Mixed"],
21    },
22    slots: {
23      type: "Number",
24    },
25    booked: {
26      type: "Number",
27    },
28    category: {
29      type: ["String"],
30    },
31    duration: {
32      type: "String",
33    },
34    level: {
35      type: "String",
36    },
37    date: {
38      type: "String",
39    },
40    time: {
41      type: "String",
42    },
43  },
44  { timestamps: true }
45 );
46
47 const Workshop = mongoose.model("Workshop", workshopSchema);

```

We wanted to create relations between users and ticket types, workshops, available spaces. Every user can book only allowed (by the ticket) number of workshops. Admin user could create, edit, and delete Workshops and Talks

HOSTING

DEPLOYMENT

Conference App is going to be hosted in the cloud on Heroku – Platform as a Service (PaaS). It allows deployment of the website just in couple steps. It is very good choice for students, start-ups and quick deployment of prototypes or proof of concepts thanks to free testing accounts.

To deploy MERN stack application we need to install Heroku command line tool and:

- login to the Heroku service - *heroku login*,
- Create an App Heroku – *heroku create cinecon2021*
- Create git repository if not created before and add all required files – *git add* .
- Commit *git commit -m "prepare for deployment"*
- Add remote for Production app *heroku git:remote -a cinecon2021*
- Push *git push heroku main*

- Inside package.json we need add script: "heroku-postbuild":
"NPM_CONFIG_PRODUCTION=false npm install --prefix frontend && npm run build --prefix frontend"
- *And app is live here <https://cinecon2021.herokuapp.com/>*

SCALABILITY

Currently, the proof of concept is hosted at Heroku.com and the associated database is hosted on a server offered by MongoDB Atlas. Both services are only sufficient to demonstrate the basic functionality of WebApps and are not sufficient to work with them on a professional level. This brings us to the issue of scaling.

Since we are offering a conference web application, we have a strong sessional peak that we will feel during the weekend of the conference and from about 2 months before. As the site operator, we can easily choose the time of the greatest database load by selecting when tickets are available for purchase and when workshops are bookable. These two functions will then generate the rush on our server capacities.

For the rest of the year, the web app will probably only have representative functions and will hardly have to carry any relevant server loads. Accordingly, we have to be flexible in order to be able to absorb the peaks.

With Heroku, it is relatively easy to switch between the booked packages. Nevertheless, you need at least the "Performance M" package to get autoscaling as well as dedicated server power. We would recommend this for the time of heavy server load, as it may be necessary to quickly absorb strong fluctuations while the conference is running and many users inform themselves about the following event on the WebApp between events. The "Performance M" package costs from \$250 per month. However, this can go up into 4 digits depending on the amount of server power needed. For hosting the backend and the API server, a vertical scaling strategy would be more appropriate, as it would not be worth the effort to develop a parallel horizontal solution. For this, the conference will be niche and thus small enough that such a strategy will not be necessary.

For the frontend and the database, however, it would be conceivable to also consider a horizontal scaling strategy, as MongoDB is inherently well suited to being distributed on multiple servers. We currently use the free MongoDB Atlas hosting, which is designed for a maximum storage capacity of up to 5GB. For our test data of 3 users we used a capacity of 649B for the data and 72KB for the indexes. For the 3 workshops we needed 9KB of data and 20KB of indexes. Extrapolating this, even with 10,000 users and 500 workshops, we would be at a data capacity in the single-digit gigabyte range. Of course, some data is still missing from our proof of concept, but presumably the data load is not the problem, but rather the frequency of requests. This is clearly limited in our current plan, since we do not get dedicated server performance, but are on a shared server. Therefore, we would recommend moving to a server that offers permanent performance (MongoDB Atlas from \$57). You would also need to keep an eye on MongoDB's monitoring tools and, just before the conference, consider whether it is worth switching to the horizontally scaled "Multi Region Dedicated Server" package from MongoDB Atlas. If the users are scattered around the globe and the data load of the normal dedicated server is not enough, this would be a convenient solution, as MongoDB is well suited to be scaled horizontally. The price for this starts at just under 100\$.

The frontend hosting is the most uncomplicated for the WebApp. It can be scaled horizontally on physical and virtual servers without any problems, because only the already optimised HTML, JS and CSS documents and a few optimised images have to be delivered to the clients. There is no server-side rendering, which could place a large load on the frontend server. Ideally, it is also ensured here that an automatically scaling option is booked from the time the workshop bookings are activated, in order to have as few downtimes as possible. Particularly when booking workshops, it is sometimes important to book quickly in order to get a place for your favourite workshop.

TRACKING AND STATISTICS

Monitoring your application can spot issues in advance and respond to incidents quickly. Heroku offers several tools which allow to monitor all aspect of Heroku apps.

- Logging tools – collect and store application and database logs. Logs can be used during incidents to help with identification of the problem and narrowing down options for fixing the problem or logs can be good source of information after accident.
- Application performance monitoring (APM) display details about apps performance, can help with identification of problematic parts which slow down the app.
- Error monitoring captures errors thrown by code of the application, dependencies, or frameworks
- Platform monitoring tool can capture router metrics and present it in visual dashboard, very efficient in monitoring of rapid increases in database load
- Heroku threshold alerting gives you notification when response times or error rate gets to high. Heroku's recommended response time is 500ms or less.
- Heroku Status provides updates on maintenance event and platform incidents. Maintenance alerts allow to prepare in advance.

CONCLUSIONS

Our web app is on track to meet the requirements for a modern online conference system. We have thought carefully in advance about the target group we want to address and how we can achieve the best possible experience for them. By breaking it down into bookable workshops and the inclusive talks, we have both created a reasonable pricing structure and ensured that conference participants can specialise and get as much out of the conference as possible in small group sessions.

Technologically, we tried to build a web app that runs as fast as possible, is easily scalable and up to date, is easily expandable due to its structure and has the potential to be used for other conferences in the future.

In the end, due to personal setbacks, the proof of concept did not achieve the quality and functionality we had originally envisaged. Nevertheless, it shows that our plan is working and that it would have been possible to realise our goals if we had had more time.

BIBLIOGRAPHY

Hoque, S., 2020. *Full-Stack React Projects*. 2 ed. Birmingham: Packt Publishing.

J., C., 2013. *Mobile App development, JavaScript everywhere and "the three amigos"*, New York: IBM.

OpenJS Foundation, 2020. *Express*. [Online]

Available at: <https://expressjs.com/>

[Accessed 3 12 2020].

reactjs.org, 2020. *JSX in Depth*. [Online]

Available at: <https://reactjs.org/docs/jsx-in-depth.html>

[Accessed 1 12 2020].

Subramanian, V., 2019. *Pro MERN Stack*. 2 ed. Bangalore: Apress.

V., S., 2019. *Pro MERN stack*. 2nd ed. Bangalore: Apress.

Wilson, J. R., 2018. *Node.js & the Right Way*. 1 ed. Raleigh: Andy Hunt - The Pragmatic Bookshelf.