FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Web Applications
2018/2019

Project report

Group members:

54842, Ondřej Zemánek
54888, Hubert Kamil Olkiewicz
Duarte Moreira

December 20, 2018

1. Project goals
    - Scenario analysis in order to create user stories
    - Based upon user stories, implementation of backend server
    - IFML diagram generation
    - IFML diagram, implementation of frontend application
2. Work Contribution
    - **Hubert Olkiewicz**
      - Project initialization (*Java*), security configuration and implementation, swagger configuration (swagger description file).
      - Repositories, controllers and services for: *authentication*, *user*, *company*, *eventproposal*.
      - Tests: two exemplary test of services, one written in Java using *Mockito*, second *Groovy* + *Spock*.
      - Project initialization (*React*) and configuration (*package.json*, *tslinter*, *tsconfig*), secured routes, token management.
      - Views: *Login*, *EventProposal, Comments, EventProposalReviews* fixes and adjustments in all views
    - **Ondřej Zemánek**
      - Data model
      - repositories, controllers and services for: Comment, Company EventProposal, EventProposalReview
      - email notifications
      - User stories
      - IFML diagram
      - FrontEnd - Header, Companies, Users, My profile, Homepage, Styling, Event proposals refactoring, proposal approval
    - **Duarte Moreira**

3. System description

   3.1. Scenario interpretation

Marketing company called ECMA needs application to host and organise events with the help of partner companies. Members of ECMA are users of the system.The system also holds information about the partner companies and their employees (partner users) and allows the corresponding users to maintain their data. Users from a given company tagged as "Partner Admin" can edit the information of the company, and manage the users from that same partner company. Any user (even an employee from a partner company) may introduce a new event proposal in the system. A proposal involves members of the host company (users) and members of the partner company (partner admins and partner users). A proposal goes through an approval process before becoming public to all users. The reviewing is performed by adding reviews (comments) to the event proposal, the assignment of reviewers to proposals is a bidding process, the approval is performed by a staff user tagged "Approver". A proposal consists of an arbitrary sequence of document sections (e.g. title, description, goals, needed materials, budget, work-plan). A proposal has a team that consists of users and partner members. A proposal has a related stream of comments that fire email notifications to the involved users to respond using a link.

3.2. Technologies used
- Database: H2 database
- Backend server: Java11 + Springboot 2.1.0
- Frontend: React 16.6.3 + Typescript

3.3. Scenario interpretation problems
-

4. User stories

From our user stories we could construct IFML model, that will represent ours components in React frontend application.

User Stories

IMPORTANT NOTES:
ECMA_USER = any user from ECMA, that means also **ECMA_ADMIN.**
PARTNER_USER = any user from partner company, that means also **PARTNER_ADMIN logged user** = ECMA_USER or PARTNER_USER

Header:

As **Logged user**, I select **Home page** from app header to go to **Home page**.
As **Logged user**, I select **Event Proposals** from app header to go to **EventProposal page**.
As **Logged user**, I select **My Profile** from app header to go to **My Profile page**.
As **Logged user**, I select **Logout** from app header to **logout** from app and see** login page**.**
As **PARTNER_USER, I select** Company** from app header to go to **Company page**.
As **ECMA_USER**, I select **Assigned reviews** from app header to go to **Assigned reviews page**.
As **ECMA_ADMIN**, I select **Companies** from app header to go to **Companies page**.
As **ECMA_ADMIN**, I select **Users** from app header to go to **Users page**.

Login:

As **not logged user**, I open its Login page, **sign in** and see **home page**.

Home/Event:

As **PARTNER_USER**, I open its home page, select an Event from **list of most recent events** to see **event detail**, where can also see **comment stream** for this event.

My Profile:

As **logged user**, I open its My profile page so that I can see my **user detail**.
As **logged user**, I open its My profile page, select **"Edit" action** to go to **edit user form**.
As **logged user**, I open its **edit user form**, change my name, phone, email or password to see my edited info on My Profile page.
As **logged user**, I open its My Profile page, select **"delete account" action** to **delete my account**.

Company:

As **PARTNER_ADMIN**, I open its Company page to see my **company detail**.
As **PARTNER_ADMIN**, I open its Company page, select **"Edit" action** to open **edit company form**.
As **PARTNER_ADMIN**, I open its **Edit company page**, change **company name or address** and see the changed detail on **company detail**.
As **PARTNER_ADMIN**, I open its Company page, **select user** from **company users list** to see his **user detail**.
As **PARTNER_ADMIN**, I open its Company page, select **"add new user" action** to open **add user form**, where I can create only partner user or partner admin, to **create new partner user or partner admin for my company** and see him in User list on My Company page.

AssignedReviews:

As **ECMA_USER**, I open Assigned reviews page, select **Event proposal** from **list of assigned eventProposal to review** to go to **EventProposal detail page**.

Users:

As **ECMA_ADMIN**, I open its users page, select **user** from **list of all users** to see **user detail**.
As **ECMA_ADMIN**, I open its **user detail**, select **"Delete" action** to **delete user account**.
As **ECMA_ADMIN**, I open its **user detail**, select **"Edit" action** to open **user edit form** to edit user detail.
As **ECMA_ADMIN**, I open its **user edit form**, change any user detail to see changed user in Users list on Users page.
As **ECMA_ADMIN**, I open its users page, select **"create new user" action** to see **add user form**, where I can create user with any role available, to **create new user with any role** and see him in User list on Users page.

Companies:

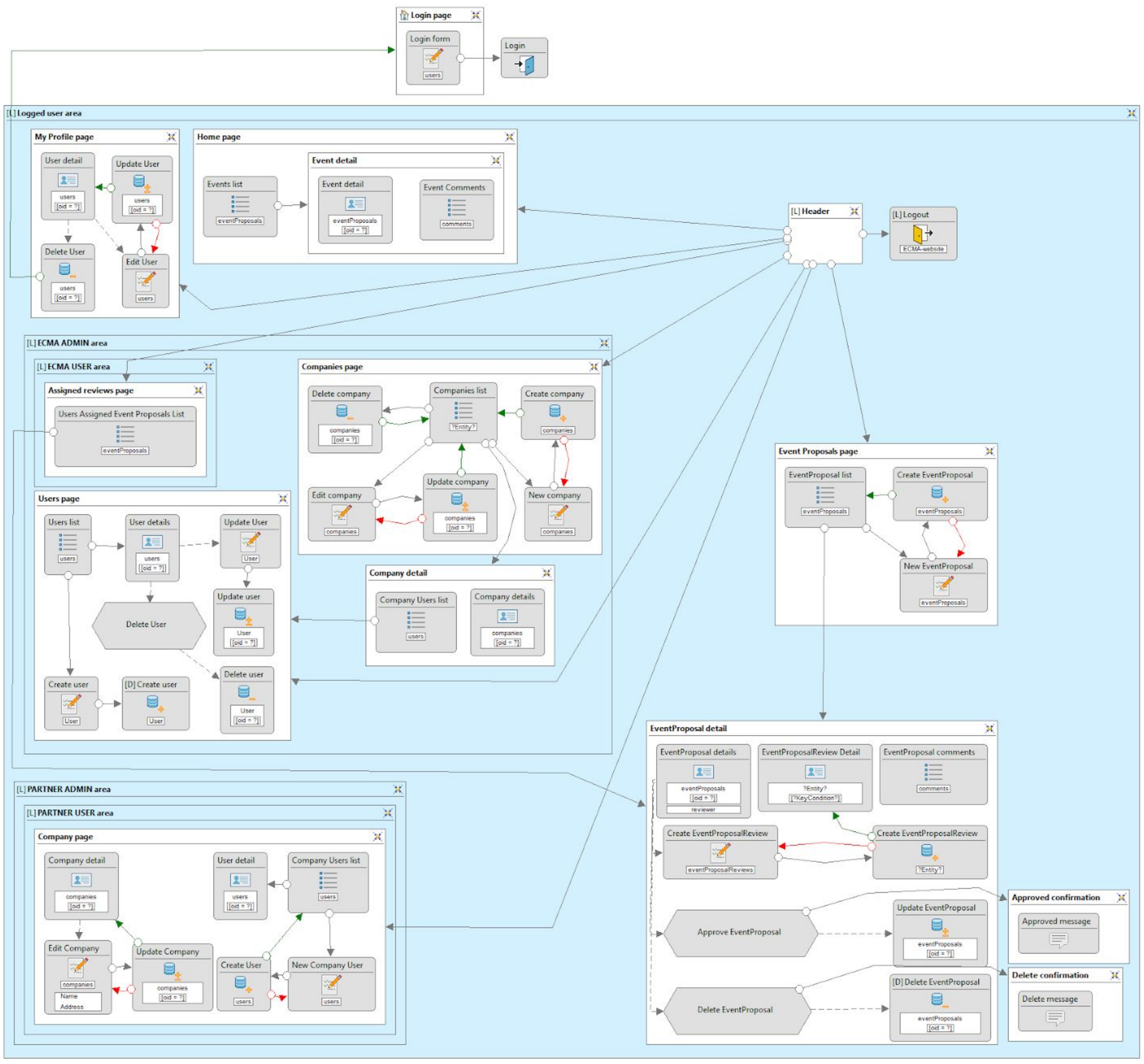
As **ECMA_USER**, I open its Companies page, select **company** from **list all companies** to see **company detail and company users list**.
As **ECMA_ADMIN**, I open its Companies page, select **company** from **list all companies**, select user from **company users list** to see **user detail** on Users page.
As **ECMA_ADMIN**, I open its Companies page, select **company** from **list all companies**, select **edit action** to edit company detail in **company edit form**.

As **ECMA_ADMIN**, I open its Companies page, select **company** from **list all companies**, select **delete action** to delete company with all company users.
As **ECMA_ADMIN**, I open its Companies page, select **"create new" action** to open **add company form**. As **ECMA_ADMIN**, I complete its **add company form** to see new company in companies list on companies page.

Event Proposals:

As **logged user**, I open its Event Proposals page, select **Event proposal** from **event proposals list**, that I can see **Event proposal detail**, where I also see **comment stream** for this event proposal.
As **logged user**, I open its **Event proposal detail** to see also Event proposal reviewer if it is already available.
As **logged user**, I open its Event Proposals page, select **"Create new" action** to see **Add Event proposal form**.
As **logged user**, I open its **Add Event proposal form**, complete it and see it in event proposal list on Event Proposal page.
As **user with Proposal approvement right**, I open its **Event proposal detail** of event proposal which has review, select **"Approve"** to **approve event proposal**, see it on Home page in **Events list**.
As **ECMA_ADMIN**, I open its **Event proposal detail**, select **"Delete" action** to **delete event proposal** and see **updated proposal list** on Event Proposal page.
As **ECMA_USER**, I open its **Event proposal detail** of event proposal without assigned reviewer, select **"Assign me" action*** to assign myself as reviewer.
As **ECMA_ADMIN**, I open its **Event proposal detail** of event proposal without reviewer, select **"assign user"**, select user from **list of users for reviewing** to **assign reviewer for event proposal**.
As **ECMA_USER**, I open **Event proposal detail** of event proposal, that I am assigned to review to open **Add review form**.
As **ECMA_USER**, I open **Add review form**, complete it to see Review on **Event proposal detail**.
As **logged user**, I open its **Event proposal detail** of event proposal I am involved in to see **event proposal discussion**.
As **logged user**, I open its **Event proposal detail** of event proposal I am involved in, select **"Add comment" action** in **discussion**, write comment to see my new comment in discussion.
As **ECMA_ADMIN**, II open its **Event proposal detail**, select **"delete comment" action** on comment in **discussion** to **delete comment**.

5. IFML diagram

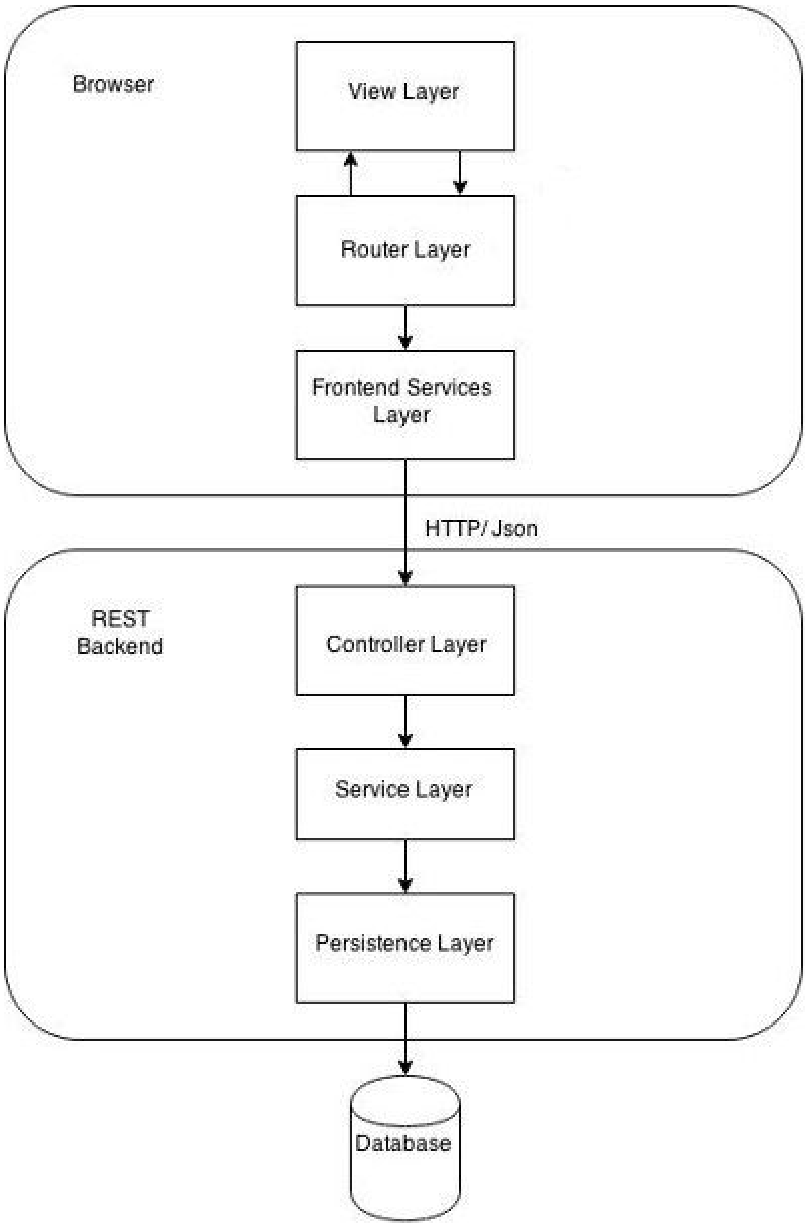


IFML Diagram (*ifml-diagram.png*)

6. Architecture

As application architecture we have used nowadays standard architectural pattern which separates business layer from view layer and communicates with each other through HTTP requests.
Browser component is implemented using Typescript and React framework which by state management through user actions, proper views are shown.
REST Backend component is implemented using Java11 and Springboot. It is divided into Controller layer, Service layer and Persistence Layer which uses

H2 database, but that can be easily switched to another OLTP database type. Controller maps JSON request into Java objects and passes it to Service layer which is responsible for business logic. After successfully confronting business logic, data is persisted in database and response is sent to browser.

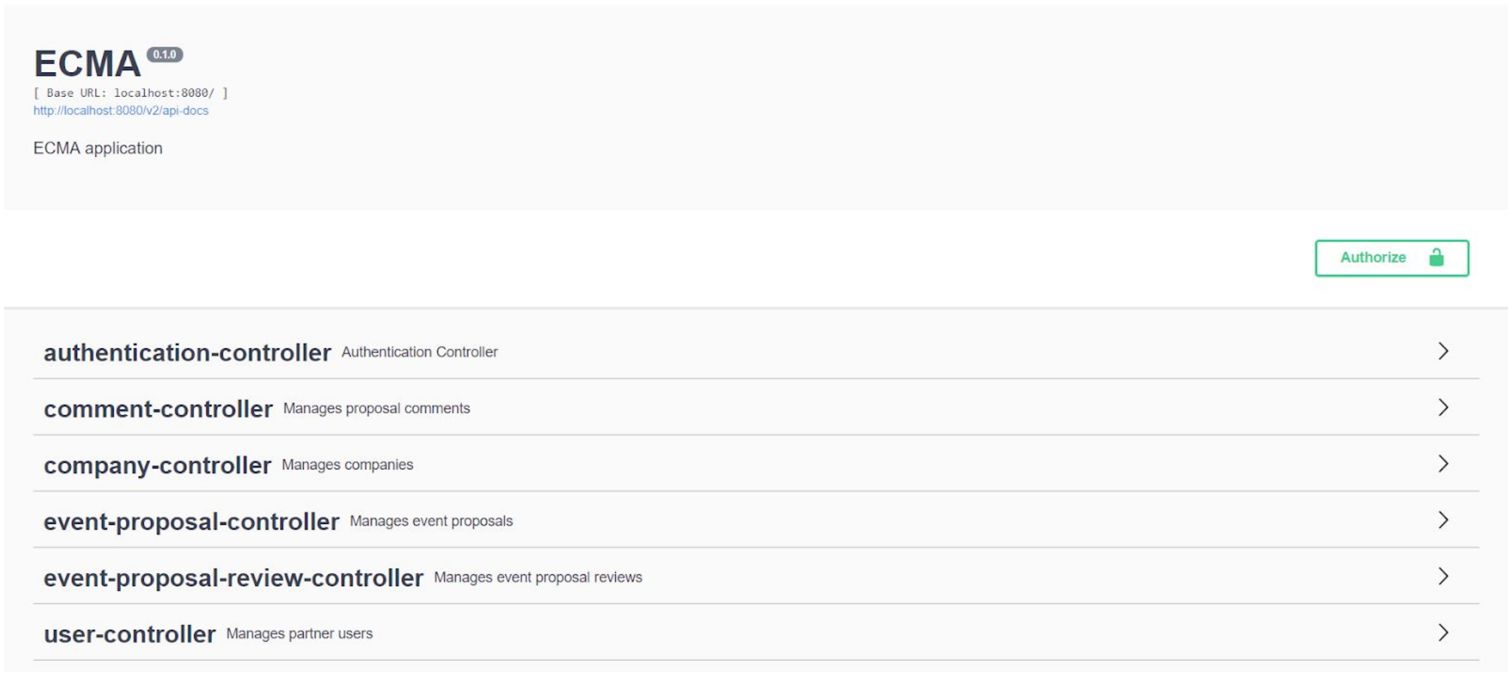


Architecture depiction

7. Backend

7.1. Controller layer

7.1.1. Swagger view


ECMA 0.1.0

[Base URL: localhost:8080/]

http://localhost:8080/v2/api-docs

ECMA application

Authorize

authentication-controller Authentication Controller

comment-controller Manages proposal comments

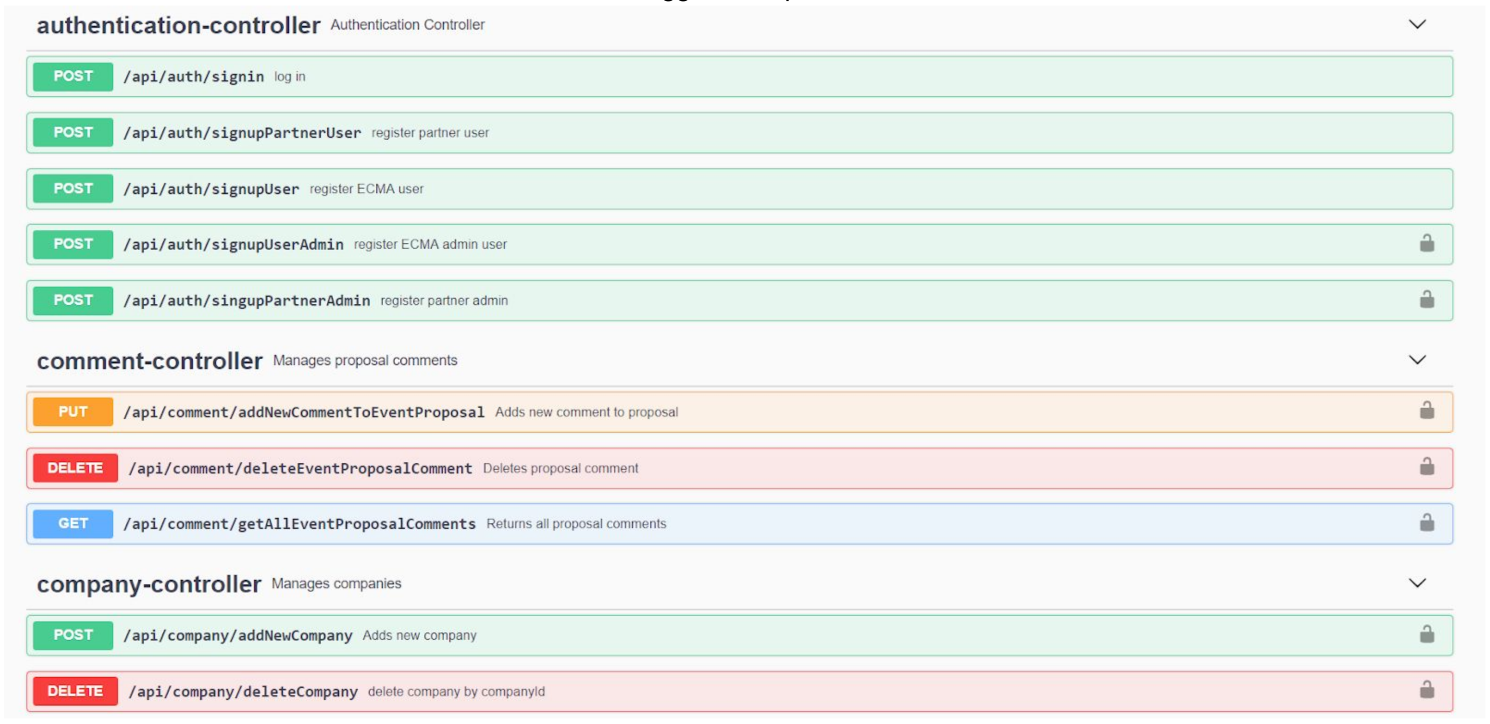
company-controller Manages companies

event-proposal-controller Manages event proposals

event-proposal-review-controller Manages event proposal reviews

user-controller Manages partner users

Swagger, collapsed view


authentication-controller Authentication Controller

POST /api/auth/signin log in

POST /api/auth/signupPartnerUser register partner user

POST /api/auth/signupUser register ECMA user

POST /api/auth/signupUserAdmin register ECMA admin user

POST /api/auth/singupPartnerAdmin register partner admin

comment-controller Manages proposal comments

PUT /api/comment/addNewCommentToEventProposal Adds new comment to proposal

DELETE /api/comment/deleteEventProposalComment Deletes proposal comment

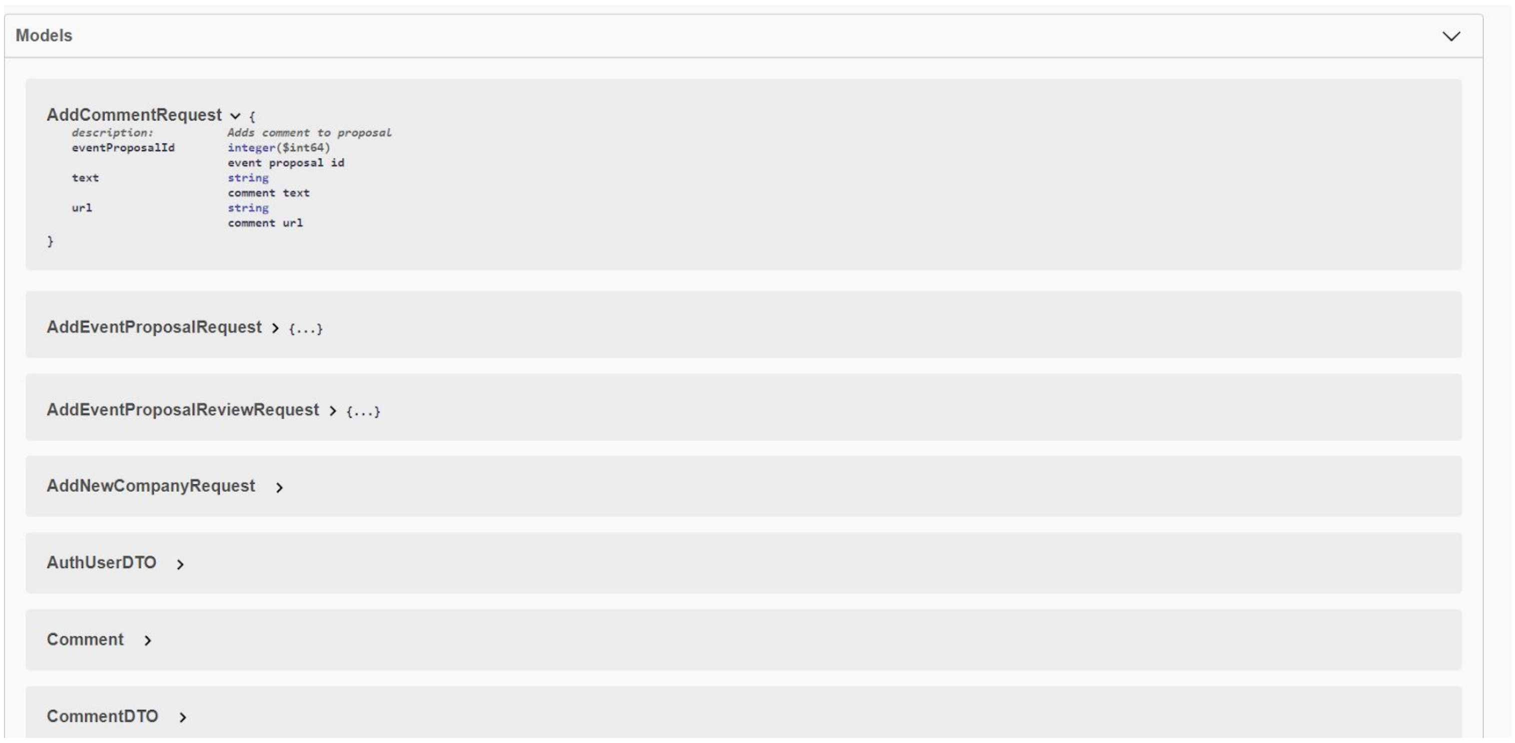
GET /api/comment/getAllEventProposalComments Returns all proposal comments

company-controller Manages companies

POST /api/company/addNewCompany Adds new company

DELETE /api/company/deleteCompany delete company by companyId

Swagger api view



```
Models

AddCommentRequest ⌄ {
    description:        Adds comment to proposal
    eventProposalId     integer($int64)
                        event proposal id
    text                string
                        comment text
    url                 string
                        comment url
}

AddEventProposalRequest › {...}

AddEventProposalReviewRequest › {...}

AddNewCompanyRequest ›

AuthUserDTO ›

Comment ›

CommentDTO ›
```

Swagger, data models

Swagger configuration class:

```java
@Configuration
@EnableSwagger2WebMvc
@Import(SpringDataRestConfiguration.class)
public class SwaggerConfig {
   private static final String API_KEY_NAME = "Token /auth/signin. Add Bearer
before TOKEN";

   @Bean
   public Docket api() {
       return new Docket(DocumentationType.SWAGGER_2)
               .ignoredParameterTypes(LoggedUser.class)
               .securitySchemes(getSecuritySchemes())
               .securityContexts(getSecurityContexts())
               .apiInfo(apiInfo())
               .select()
               .apis(RequestHandlerSelectors.any())
               .paths(PathSelectors.ant("/api/**"))
               .build();
   }


private ArrayList<ApiKey> getSecuritySchemes() {
   return newArrayList(new ApiKey(API_KEY_NAME, "Authorization", "header"));
}

private ArrayList<SecurityContext> getSecurityContexts() {
   return newArrayList(SecurityContext.builder()
           .securityReferences(newArrayList(getSecurityReference()))
           .forPaths(getSecuredEndpointsPredicate())
           .build());
}

private Predicate<String> getSecuredEndpointsPredicate() {
   return path -> {
       AntPathMatcher matcher = new AntPathMatcher();
       return !matcher.match("/api/auth/**", path) ||
matcher.match("/api/auth/signupUserAdmin", path) ||
matcher.match("/api/auth/singupPartnerAdmin", path);
   };
}

private SecurityReference getSecurityReference() {
   return SecurityReference.builder()
           .reference(API_KEY_NAME)
           .scopes(getAuthorizationScopes())
           .build();
}

private AuthorizationScope[] getAuthorizationScopes() {
   List<AuthorizationScope> authScopeList = new ArrayList<>();
   authScopeList.add(new
AuthorizationScopeBuilder().scope("global").description("token access
needed").build());
   return authScopeList.toArray(new AuthorizationScope[0]);
}
```

7.1.2. Exemplary controller class

Below we can see EventProposalController implementation. We have used Swagger API to describe and create *swagger-ui.html* file, by annotating controller as well as all classes which were used as request bodies (DTO classes).
Two exemplary methods are exposed. One regarding updating event proposal which is additionally secured by role check. Second one gets all data using paging repository interface to get only desired amount of records.

```
@ApiOperation(httpMethod = "PATCH",
       value = "update proposal",
       nickname = "updateEventProposal")
@PatchMapping("updateEventProposal")
@Secured({RoleName.USER, RoleName.ADMIN})
public ResponseEntity updateProposal(@LoggedUser AuthUser loggedUser,
@RequestBody UpdateEventProposalRequest request) {
   try {
       eventProposalService.updateEventProposal(loggedUser, request);
   } catch (BadRequestException e) {
       return new ResponseEntity<>(e.getMessage(), HttpStatus.BAD_REQUEST);
   }
   return new ResponseEntity(HttpStatus.OK);


@ApiOperation(httpMethod = "GET",
       value = "get all proposals",
       nickname = "getAllProposals")
@GetMapping("getAllProposals")
public ResponseEntity<GetAllProposalsResponse> getAllProposals(@RequestParam int
page, @RequestParam int pageSize) {
   Page<EventProposal> pageResponse =
eventProposalRepository.findAll(PageRequest.of(page, pageSize));
   List<EventProposalDTO> eventProposalDTOList = pageResponse.stream()
           .map(eventProposal -> mapper.map(eventProposal,
EventProposalDTO.class))
           .collect(Collectors.toList());
   return new ResponseEntity<>(
           new GetAllProposalsResponse(eventProposalDTOList,
pageResponse.getTotalPages()),
           HttpStatus.ACCEPTED);
}
```

7.2. Service layer

Services are responsible for business logic layer (business decisions and validations, interacting with database). Below exemplary implementation of EventProposalService. AddEventProposal, adds new event proposal and also connects user company as involved one to proposal so everyone from this company might comment this event proposal. UpdateEventProposal updates attribute values of exact proposal. As we can see, user has to be approval to be able to modify data. It is an example of business logic.

```
@Service
@RequiredArgsConstructor
public class EventProposalService {
   private final EventProposalRepository eventProposalRepository;
   private final UserRepository userRepository;
   private final UserService userService;

   @Transactional
   public void addNewEventProposal(AuthUser loggedUser, AddEventProposalRequest
request) {
       EventProposal event = createEventProposal(loggedUser, request);
       if (UserUtils.isPartnerUser(loggedUser)) {
     event.setInvolvedCompany(loggedUser.getUser().getCompany());
       }
       eventProposalRepository.save(event);
   }

   @Transactional
   public void updateEventProposal(AuthUser loggedUser,
UpdateEventProposalRequest request) {
       EventProposal eventProposal =
eventProposalRepository.findById(request.getId())
               .orElseThrow(() -> new BadRequestException("Non existing
proposal"));
       if (loggedUser.getUser().getIsApproval()) {
           eventProposal.setApproved(request.isApproved());
           eventProposal.setBudget(request.getBudget());
           eventProposal.setDescription(request.getDescription());
           eventProposal.setGoals(request.getGoals());
           eventProposal.setWorkPlan(request.getWorkPlan());
           eventProposal.setNeededMaterials(request.getNeededMaterials());
           eventProposal.setTitle(request.getTitle());
       } else {
           throw new BadRequestException("User is not assigned to event
proposal");
       }
   }
```

7.3. Persistence layer

Persistence layer is divided into Classes describing database tables (attributes, relationships etc.) and Repositories which makes us able to operate on database, using easy interface naming conventions which at compilation time are transformed into HQL queries.

Exemplary entity:

```
@Entity
@Table(name = "event_proposals")
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class EventProposal {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   @NotBlank
   @Size(max = 40)
   private String title;

   @Size(max = 500)
   private String description;

   // rest of attributes ...

   @ManyToOne
   private Company involvedCompany;

   @ManyToOne
   private User assignedReviewer;

   @OneToOne(mappedBy = "eventProposal", cascade = CascadeType.ALL,
orphanRemoval = true)
   private EventProposalReview review;

   @OneToMany(mappedBy = "eventProposal", cascade = CascadeType.ALL,
orphanRemoval = true)
   private List<Comment> comments = new ArrayList<>();
```

Exemplary Repository:

```
@Repository
public interface CommentRepository extends JpaRepository<Comment, Long> {
   List<Comment> findByEventProposalOrderByTimestampDesc(EventProposal
eventProposal);
   List<Comment> findByEventProposalAndAuthor(EventProposal eventProposal, User
author);
   List<Comment> findByAuthor(User author);
   boolean existsByEventProposalAndAuthor(EventProposal eventProposal, User
author);
}
```

7.4. Security

7.4.1. Spring configuration

Spring offers already configurable security mechanisms. simplest approach is to create class which extends WebSecurityConfigurerAdapter which has three mandatory methods to override: configure(AuthentitacionManagerBuilder), authentitactionManagerBean() and configure(HttpSecurity).

configure function regards passing to spring services which are responsible for user authentication:

```
@Override
public void configure(AuthenticationManagerBuilder authenticationManagerBuilder)
throws Exception {
   authenticationManagerBuilder
           .userDetailsService(customUserDetailsService)
           .passwordEncoder(passwordEncoder());
}
```

authenticationManagerBean() function requires to pass AuthenticationManager object which later on is used by Spring:

```
@Bean(BeanIds.AUTHENTICATION_MANAGER)
@Override
public AuthenticationManager authenticationManagerBean() throws Exception {
   return super.authenticationManagerBean();
}
```

configure(HttpSecurity) function is responsible for configuration related to HTTP protocol like managing permitted and authenticated endpoints. Also filter was added to retrieve requesting user from his JWT token at runtime (where at controllers adding only *@LoggedUser* annotation was enough to get this user entity from database)

```
 @Override
 protected void configure(HttpSecurity http) throws Exception {
    http
            .cors()
            .and()
            .csrf().disable()
            .headers()
            .frameOptions().disable()
            .and()
            .exceptionHandling().authenticationEntryPoint(unauthorizedHandler)
            .and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and()
            .authorizeRequests().antMatchers("/",
            "/favicon.ico", "/**/*.png", "/**/*.gif", "/**/*.svg", "/**/*.jpg",
"/**/*.html", "/**/*.css", "/**/*.js")
            .permitAll()
            .antMatchers("/h2-console/**")
            .permitAll()
            .antMatchers("/api/auth/**")
            .permitAll()
            .antMatchers("/swagger-resources/**", "/swagger-ui.html",
"/v2/api-docs", "/webjars/**")
            .permitAll()
            .anyRequest().authenticated();

    // Add our custom JWT security filter
    http.addFilterBefore(jwtAuthenticationFilter(),
 UsernamePasswordAuthenticationFilter.class);
 }
```

7.4.2. Implementation

All tables and functionalities regarding security and authorization have to be implemented (like *CustomUserDetailsService* which is visible in configure method). Our implementation was separated into two major parts. One managing authorized user account table (entity, repository and service) and second was responsible for JWT token functionality.
On top of these two parts we built AuthenticationController which responsibility was to register and authorize users by handling and reading tokens.

7.5. Tests

Exemplary tests were written, one using Java and Mockito framework, second using Groovy and Spock framework. It’s worth to note that Groovy with Spock can also work really well with Mockito (since Groovy is compiled to Java bytecode). The main reason why Groovy + Spock was used as an example is that it simplifies a lot more complex checks, introduces data tables (so you don’t have to bother anymore with inconvenient *ParametrizedTest*) or simple Collections initialization (i.e *def map = [key1:val1, key2:val2])* which shortens code.

In addition multiple manual tests were conducted.

Exemplary Java + Mockito test:
It is testing whether partner user data was successfully (and properly) changed by admin user.

```
@RunWith(MockitoJUnitRunner.class)
public class UserServiceTest {

   @Mock
   AuthUserRepository authUserRepository;
   @Mock
   CompanyRepository companyRepository;
   @Mock
   PasswordEncoder passwordEncoder;

   @InjectMocks
   UserService testedObj;

   @Test
   public void partnerUserDataShouldBeChanged() {
       //before
       var adminUser = createAdminUser();
       var oldCompany = Company.builder().name("oldCompanyName").employees(new
HashSet<>()).build();
       var newCompany = Company.builder().name("newCompanyName").employees(new
HashSet<>()).build();
       var user =
User.builder().firstname("oldFirstname").surname("oldSurname").phone("00000").co
mpany(oldCompany).build();
       var authUser =
AuthUser.builder().email("oldEmail").username("oldUsername").password("oldPasswo
rd").user(user).build();
       user.setAuthUser(authUser);
       var request = new ManagePartnerUserRequest("oldCompanyName",
               "oldUsername", "newUsername", "55555",
               "newSurname", "newCompanyName", "newEmail",
               "password", "newFirstname", false);

       //when

Mockito.when(authUserRepository.findByUsername(request.username)).thenReturn(Opt
ional.of(authUser));
Mockito.when(companyRepository.findByName(request.getNewCompany())).thenReturn(O
ptional.of(newCompany));

Mockito.when(passwordEncoder.encode(request.getNewPassword())).thenReturn("passw
ord");
       testedObj.manageUserAccountAsAdmin(adminUser, request);

       //then
       Assert.assertEquals("newCompanyName", user.getCompany().getName());
       Assert.assertEquals("newEmail", authUser.getEmail());
       Assert.assertEquals("55555", user.getPhone());
       Assert.assertEquals("newSurname", user.getSurname());
       Assert.assertEquals("password", authUser.getPassword());
   }

   private AuthUser createAdminUser() {
       var adminUser = new AuthUser();
       var adminRole = new Role();
       adminRole.setName(RoleName.ROLE_ADMIN);
       adminUser.setRoles(new HashSet<>() {{
           add(adminRole);
       }});
       return adminUser;
   }
}
```

Exemplary test Groovy + Spock:
This test checks if new event was properly added. We can see that code is much simplified (by shorter object definitions and initializations). In *then:* section we can see argument interception which takes exactly one line to create where while using mockito code you have to use distinct *ArgumentCaptor* class and configure it.

```
class EventProposalServiceTest extends Specification {

   EventProposalRepository eventProposalRepository = Mock()
   EventProposalService testedObj

   def "Should add new event with partner company"() {
       given: "Prepare tested object and test data"
           prepareTestedObj()
           AuthUser authUser = createPartnerCompanyUser()
           def request = new AddEventProposalRequest()
           request.title = "title"

       when:
           testedObj.addNewEventProposal(authUser, request)

       then: "new event with partner company should be saved"
           eventProposalRepository.save(_) >> { arguments ->
               final EventProposal eventProposal = arguments[0]
```

```
                assert !eventProposal.approved
                assert eventProposal.title == "title"
                assert eventProposal.involvedCompany == authUser.user.company
            }
    }

    @Ignore
    def prepareTestedObj() {
        testedObj = new EventProposalService(eventProposalRepository)
    }

    @Ignore
    def createPartnerCompanyUser() {
        def user = new User()
        def authUser = new AuthUser()
        user.authUser = authUser
        authUser.user = user
        def adminRole = new Role()
        adminRole.setName(RoleName.ROLE_PARTNER_USER)
        authUser.setRoles([adminRole] as Set)
        def company = new Company()
        user.company = company
        company.employees = [user] as Set
        authUser
    }
}
```

8. Frontend

8.1. Typescript configuration

Typescript compiler can be configured to drop some rules which in our case were inconvenient and really time consuming while developing application like for an instance *forceConsistentCasingInFileNames* or *noUnusedLocals.*

Also we have used *Tslinter* for static code analysis with three default extensions *tslint:recommended,tslint-react, tslint-config-prettier*, but as well in this case we had to drop few rules to speed up development (like o*rdered-imports* or *no-console*).

8.2. Services and View layer

Service functionality was incorporated into View components because mostly it was responsible only for HTTP requests to backend server so for the sake of simplicity (with dealing with asynchronous HTTP calls) we merged them.

View in react operates on *props* and *state* and related to their changes component callbacks (like *ComponentDidMount*). Based upon this we can create reactive, single page view.

Exemplary View:
As we can see, our class extends *React.Component* to be able to cooperate with

React environment. We have created interfaces describing *state* and *props* of component. This view was rendering modal with event attributes to be updated. Two callbacks were required, *onClose* and *onUpdate* to communicate with parent component. There is also *updateEventProposal* method which acts like service layer and it gets data from backend server.
_isMounted variable had to be introduced because of memory leak which was occuring after closing modal with update because of asynchronous HTTP method was called with *setState* which sometimes was triggered after component closure.

```
interface EventUpdateModalState {
   modalIsOpen: boolean,
   titleInput: string,
   ...
}

interface EventUpdateModalProps {
   eventProposal: EventProposalDTO;
   onCloseCallback?(): void,
   onUpdateCallback?(updatedEventProposal: EventProposalDTO): void,
}

export default class EventUpdateModal extends
React.Component<EventUpdateModalProps, EventUpdateModalState> {
   /* tslint:disable */
   public _isMounted = false;
   /* tslint:enable */

   constructor(props: EventUpdateModalProps) {
       super(props);
       this.state = {
           ...state initailization with default values
       }
   }

   public componentDidMount() {
       this._isMounted = true
   }

   public componentWillUnmount() {
       this._isMounted = false
   }

   public render() {
       return (
           <Modal isOpen={this.state.modalIsOpen} toggle={this.closeModal}>
               <ModalHeader
toggle={this.closeModal}>{this.props.eventProposal.title}</ModalHeader>
               <ModalBody>
                   <FormGroup>
                       <FormGroup>
                           <Label for="title">Title</Label>
```

```
                            <Input type="text" id="title" placeholder="Title"
                                   value={this.state.titleInput}
                                   onChange={e => this.setState({titleInput:
e.target.value})}/>
                        </FormGroup>
              //...form  inputs...
                      </ModalBody>
               <ModalFooter>
                   <Row>
                       <Col>
                           <Button onClick={this.updateEventProposal}
color="success">Update</Button>
                       </Col>
                       <Col>
                           <Button onClick={this.closeModal}
color="danger">Cancel</Button>
                       </Col>
                   </Row>
               </ModalFooter>
           </Modal>
       );
   }


   private updateEventProposal = () => {
       console.info(this.mapStateToEventProposalDTO())
       const instance = axios.create();
       instance.patch(`/api/eventProposal/updateEventProposal`,
           {...this.mapStateToEventProposalDTO()}
       )
           .then(response => {
               if (this.props.onUpdateCallback) {

this.props.onUpdateCallback(this.mapStateToEventProposalDTO());
               }
               this.toggleModal();
           })
           .catch(reason => {
               this.toggleModal();
               console.error(reason);
           })
   };
```

8.3. Router

We used *react-router-dom* library for routes manipulation. As well PrivateRoute component is visible which is described in next paragraph (8.4). Into router, Header component is placed as children to be rendered despite of rendered endpoint route.

```
const Routes = () =>
   <BrowserRouter>
       <div>
           <Header/>
           <Switch>
               <Route exact path="/" component={UnsecuredView}/>
               <Route path="/login" component={Login}/>
               <PrivateRoute path="/secured" component={SecuredView}/>
               <PrivateRoute path="/eproposal" component={EventProposalView}/>
               <PrivateRoute path="/users/:userId?" component={UsersView}/>
               <PrivateRoute path="/user/new" component={UserForm}/>
               <PrivateRoute path="/user/edit/:userId" component={UserForm}/>
               <PrivateRoute path="/companies" component={CompanyView}/>
               <PrivateRoute path="/company/new" component={CompanyForm}/>
               <PrivateRoute path="/company/edit/:companyId"
component={CompanyForm}/>
               <PrivateRoute path="/profile" component={ProfileView}/>
           </Switch>
       </div>
   </BrowserRouter>;
```

8.4. Security

Two major functionalities were implemented. One responsible for redirecting from secured endpoints to login and another for token management. HTTP client (axios) was preconfigured based using token management service.

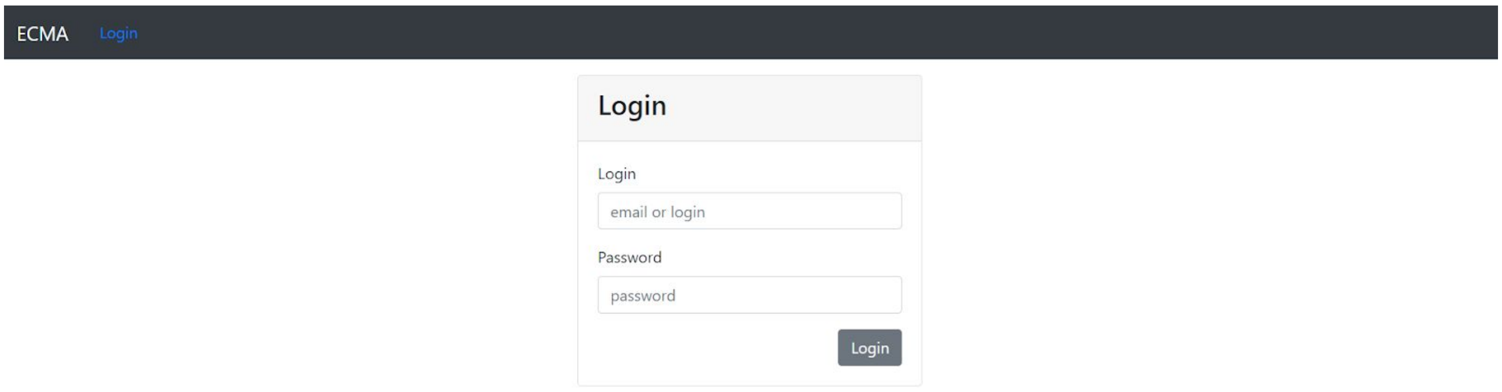
Private route implementation:
This implementation envelops standard *Route* component and based upon token existence, component is rendered or user is redirected to */login* endpoint.

```
const PrivateRoute = ({component: Component, ...rest}: any) => (
   <Route {...rest} render={(props) => (
       isTokenStored() ?
           <Component {...props}/> : <Redirect to='/login'/>
   )}/>
);
```

Token service consisted of methods *setToken*, *removeToken*, *getToken*, *getRequestHeaderToken*, *isTokenStored*, *decodedToken*, *hasRole* which are required for security management.

9. User interface

- Login


ECMA Login

Login

Login

email or login

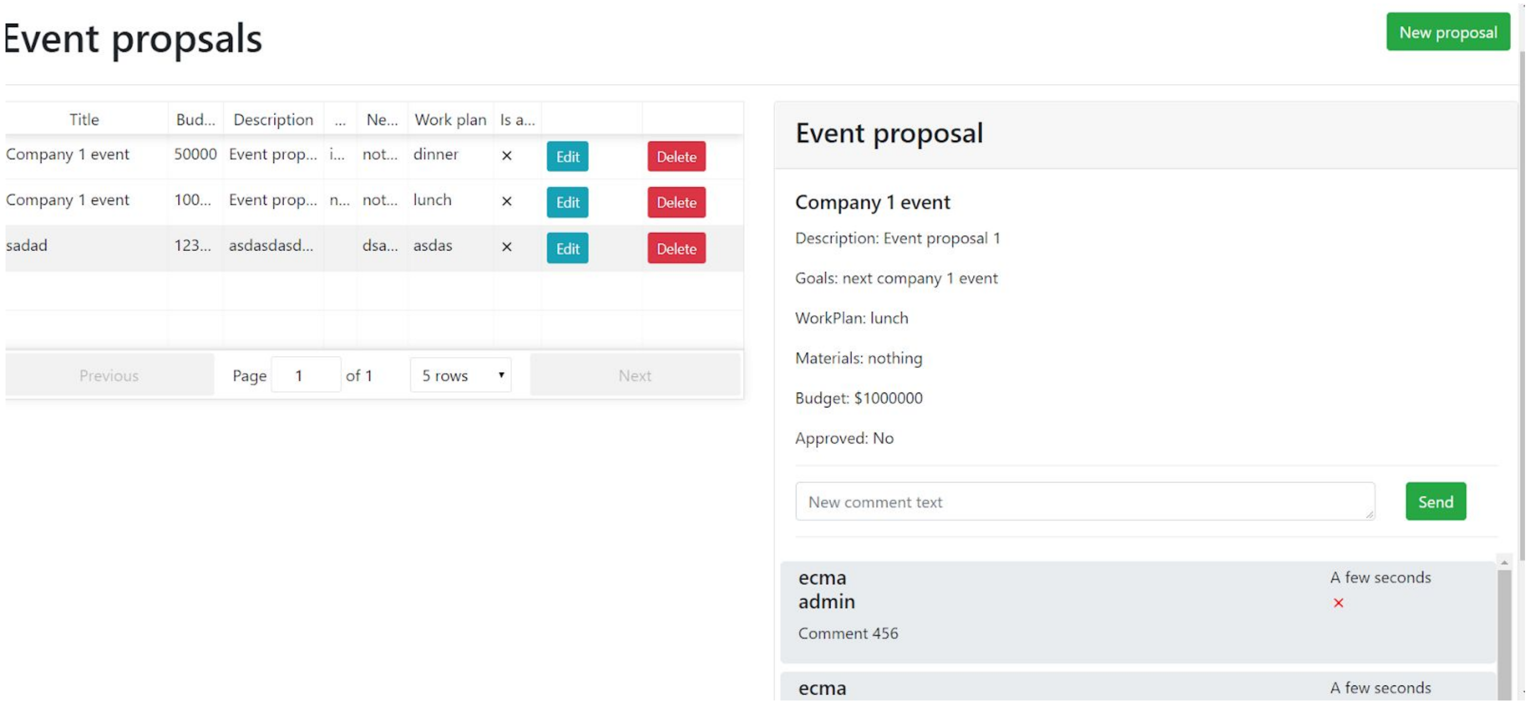
Password

password

Login

Login view

- Homepage
- Event proposals


Event propsals

New proposal

Title	Bud...	Description	...	Ne...	Work plan	Is a...		
Company 1 event	50000	Event prop...	i...	not...	dinner	×	Edit	Delete
Company 1 event	100...	Event prop...	n...	not...	lunch	×	Edit	Delete
sadad	123...	asdasdasd...		dsa...	asdas	×	Edit	Delete

Previous Page 1 of 1 5 rows Next

Event proposal

Company 1 event

Description: Event proposal 1

Goals: next company 1 event

WorkPlan: lunch

Materials: nothing

Budget: $1000000

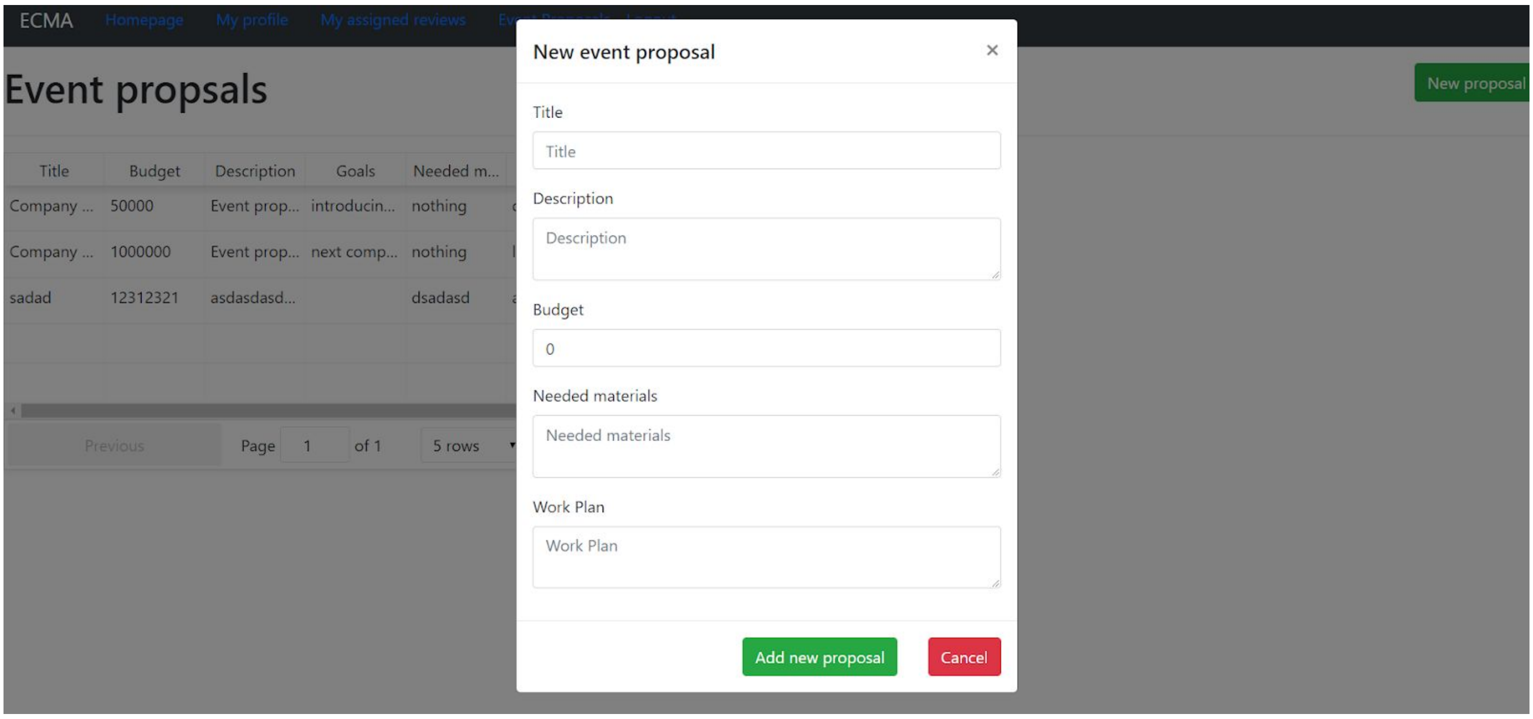
Approved: No

New comment text Send

ecma
admin
A few seconds
×
Comment 456

ecma
A few seconds

Event proposals view


ECMA Homepage My profile My assigned reviews Ev... Logout

Event propsals

New proposal

Title	Budget	Description	Goals	Needed m...
Company ...	50000	Event prop...	introducin...	nothing
Company ...	1000000	Event prop...	next comp...	nothing
sadad	12312321	asdasdasd...		dsadasd

Previous Page 1 of 1 5 rows

New event proposal ×

Title: Title

Description: Description

Budget: 0

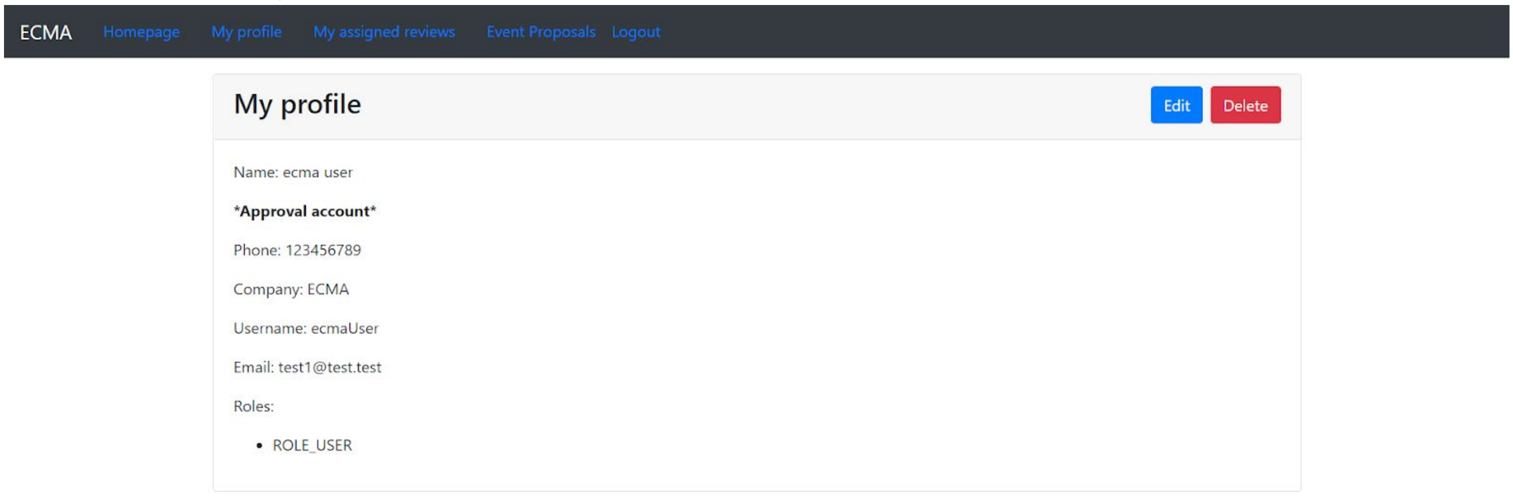
Needed materials: Needed materials

Work Plan: Work Plan

Add new proposal Cancel

New event proposal modal view

- Users
- My profile


ECMA Homepage My profile My assigned reviews Event Proposals Logout

My profile

Edit Delete

Name: ecma user

Approval account

Phone: 123456789

Company: ECMA

Username: ecmaUser

Email: test1@test.test

Roles:

- ROLE_USER

My profile view

10. Conclusions

Overall whole project wasn't hard but more likely time consuming. If you discover how to do write repository properly, writing implementation for rest of repositories is basically copying and pasting template and readjusting for current context. This applies nearly for every repeatable components like Controllers or view components (*react-table* is really good example because of variety of options and callbacks it incorporates, but once we created working, with backed server configuration, instance, basically we could just copy it into another component with small adjustments like different data fetch endpoint or cell custom view).

Particularly hardest part was understanding whole application context, dependencies responsibilities between users and their actions. Clearly writing down use cases helped a lot.

Biggest flaw of the project is lackment of JUnit tests, we have only two exemplary implemented.

Overall project really time consuming, but the time spent was worth it because of the gained experience with one of the most popular programming languages Java and Typescript as well as frameworks React and Spring, since in this particular field of web development there is still huge shortage of employees which gives us a bit upper hand on the labor market.