

Source code AuthController.js

```
1  'use strict';
2
3  const Drive = use('Drive');
4  const User = use('App/Models/User');
5
6  class AuthController {
7    async login({ request, auth }) {
8      const { username, password } = request.all();
9      const { token } = await auth.attempt(username, password);
10     const user = await User.query()
11       .profile()
12       .where({
13         username
14       })
15       .first();
16     try {
17       user.avatar = Buffer.from(await
Drive.get(`avatars/${user.avatar}`).toString('base64'));
18     } catch (error) {
19       user.avatar = Buffer.from(await
Drive.get('avatars/default.jpg')).toString('base64');
20     }
21     return {
22       user,
23       token
24     };
25   }
26 }
27
28 module.exports = AuthController;
29
```

Source code GraphQLController.js

```
1  'use strict';
2
3  const Config = use('Config');
4
5  const { graphqlAdonis } = require('apollo-server-adonis');
6  const { makeExecutableSchema } = require('graphql-tools');
7  const { fileLoader, mergeTypes, mergeResolvers } = require('merge-graphql-
  schemas');
8
9  class GraphQLController {
10   constructor() {
11     const typeDefs = mergeTypes(
12       fileLoader(Config.get('graphql.schema'), {
13         recursive: true
14       })
15     );
16     const resolvers =
17     mergeResolvers(fileLoader(Config.get('graphql.resolvers')));
18
19     this.$schema = makeExecutableSchema({
20       typeDefs,
21       resolvers
22     });
23
24     server(context) {
25       return graphqlAdonis({
26         context,
27         schema: this.$schema,
28         debug: false
29       })(context);
30     }
31   }
32
33   module.exports = GraphQLController;
34
```

Source code User.js

```
1  'use strict';
2
3  const Hash = use('Hash');
4  const User = use('App/Models/User');
5
6  module.exports = {
7    Query: {
8      async allUsers(_, { offset, limit, search }, { auth }) {
9        await auth.check();
10       const query = User.query();
11       if (search) {
12         query
13           .where('name', 'like', `%${search}%`)
14           .orWhere('username', 'like', `%${search}%`)
15           .orWhere('email', 'like', `%${search}%`);
16       }
17       if (offset) {
18         query.offset(offset);
19       }
20       if (limit) {
21         query.limit(limit);
22       }
23       return query.orderBy('name', 'asc');
24     },
25     async countAllUsers(_, { search }, { auth }) {
26       await auth.check();
27       const query = User.query();
28       if (search) {
29         query
30           .where('name', 'like', `%${search}%`)
31           .orWhere('username', 'like', `%${search}%`)
32           .orWhere('email', 'like', `%${search}%`);
33       }
34       const count = await query.count();
35       return count[0]['count(*)'];
36     },
37     async fetchUser(_, { id }, { auth }) {
38       await auth.check();
39       return User.find(id);
40     }
41   },
42
43   Mutation: {
44     async createUser(_, { name, username, email, password,
password_confirm, active }, { auth }) {
```

```

45     await auth.check();
46     if (password !== password_confirm) {
47         throw new Error('Password confirmation not match!');
48     }
49     try {
50         const user = await User.create({
51             name,
52             username,
53             email,
54             password,
55             active
56         });
57         return user;
58     } catch (error) {
59         throw new Error(`Error ${error.errno} (${error.code}): Failed to
create a new user! `);
60     }
61 },
62 async updateUser(_, { id, name, username, email, active }, { auth }) {
63     await auth.check();
64     const user = await User.find(id);
65     if (!user) {
66         throw new Error('User not found!');
67     }
68     try {
69         user.name = name;
70         user.username = username;
71         user.email = email;
72         user.active = active;
73         await user.save();
74         return user;
75     } catch (error) {
76         console.log(error);
77         throw new Error(`Error ${error.errno} (${error.code}): Failed to
update user! `);
78     }
79 },
80 async resetUserPassword(_, { id, password, password_confirm }, { auth
})) {
81     await auth.check();
82     if (password !== password_confirm) {
83         throw new Error('Password confirmation not match!');
84     }
85     const user = await User.find(id);
86     if (!user) {
87         throw new Error('User not found!');
88     }

```

```
89     user.password = await Hash.make(password);
90     await user.save();
91     return true;
92 },
93 async deleteUser(_, { id }, { auth }) {
94     await auth.check();
95     const user = await User.find(id);
96     if (!user) {
97         throw new Error('User not found!');
98     }
99     await user.delete();
100    return true;
101 }
102 }
103 };
104
```

Source code Transaction.js

```
1  'use strict';
2
3  const jwt = require('jsonwebtoken');
4  const moment = require('moment');
5
6  const Env = use('Env');
7  const { formatPages, countPages } = use('App/Helpers');
8  const Account = use('App/Models/Account');
9  const Member = use('App/Models/Member');
10 const MemberTypePrice = use('App/Models/MemberTypePrice');
11 const Transaction = use('App/Models/Transaction');
12 const KioskToken = use('App/Models/KioskToken');
13
14 module.exports = {
15   Query: {
16     async allTransactions(_, { startDate, endDate, offset, limit, orderBy
17 }, { auth }) {
18       await auth.check();
19       const query = Transaction.query()
20         .listing()
21         .has('transactionPayment');
22       if (startDate) {
23         const momentStartDate = moment(startDate);
24         if (!momentStartDate.isValid()) {
25           throw new Error('startDate is not valid datetime!');
26         }
27         query.where('created_at', '>=', momentStartDate.format('YYYY-MM-
28 DD'));
29       }
30       if (endDate) {
31         const momentEndDate = moment(endDate).add(1, 'days');
32         if (!momentEndDate.isValid()) {
33           throw new Error('endDate is not valid datetime!');
34         }
35         query.where('created_at', '<=', momentEndDate.format('YYYY-MM-
36 DD'));
37       }
38       if (offset) {
39         query.offset(offset);
40       }
41       if (limit) {
42         query.limit(limit);
43       }
44       if (orderBy) {
45         query.orderBy(orderBy.split(',')[0], orderBy.split(',')[1]);
46       }
47     }
48   }
49 }
```

```

43     }
44     return query;
45 },
46 async fetchTransaction(_, { id }, { auth }) {
47     await auth.check();
48     const transaction = await Transaction.query()
49         .listing()
50         .has('transactionPayment')
51         .where({
52             id
53         })
54         .first();
55     if (!transaction) {
56         throw new Error('Transaction not found!');
57     }
58     return transaction;
59 },
60 async countAllTransactions(_, { startDate, endDate }, { auth }) {
61     await auth.check();
62     const query = Transaction.query().has('transactionPayment');
63     if (startDate) {
64         const momentStartDate = moment(startDate);
65         if (!momentStartDate.isValid()) {
66             throw new Error('startDate is not valid datetime!');
67         }
68         query.where('created_at', '>=', momentStartDate.format('YYYY-MM-
DD'));
69     }
70     if (endDate) {
71         const momentEndDate = moment(endDate).add(1, 'days');
72         if (!momentEndDate.isValid()) {
73             throw new Error('endDate is not valid datetime!');
74         }
75         query.where('created_at', '<=', momentEndDate.format('YYYY-MM-
DD'));
76     }
77     const count = await query.count();
78     return count[0]['count(*)'];
79 }
80 },
81
82 Mutation: {
83     async createTransaction(_, { member_id, eprint_id, filename, pages }, {
request }) {
84         const token = request.header('Authorization').split(' ')[1];
85         const kioskToken = await KioskToken.query()
86             .where({

```

```

87     Token
88   })
89   .first();
90   if (!kioskToken) {
91     throw new Error('Token is invalid!');
92   }
93   const decodedToken = jwt.verify(token, Env.get('JWT_SECRET_KIOSK'));
94   const formattedPages = formatPages(pages);
95   const num_pages = countPages(formattedPages);
96   if (num_pages === 0) {
97     throw new Error('Please input valid page number!');
98   }
99   const member = await Member.query()
100     .listing()
101     .where({
102       member_id
103     })
104     .first();
105   const account = await Account.query()
106     .where({
107       member_id
108     })
109     .first();
110   if (!member || !account) {
111     throw new Error('Member not found!');
112   }
113   if (!account.active) {
114     throw new Error('Account is disabled!');
115   }
116   const memberTypeprice = await
MemberTypePrice.find(member.member_type_id);
117   if (!memberTypeprice) {
118     throw new Error('Harga untuk jenis akun Anda belum tersedia.
Silakan menghubungi petugas.');
```



```

132     throw new Error('Saldo tidak mencukupi!');
133 }
134
135 const transaction = await Transaction.create({
136     member_id,
137     kiosk_id: decodedToken.uid,
138     eprint_id,
139     filename,
140     pages: formattedPages,
141     num_pages,
142     price,
143     total_price
144 });
145 return Transaction.query()
146     .listing()
147     .where({
148         id: transaction.id
149     })
150     .first();
151 },
152 async cancelTransaction(_, { id }, { request }) {
153     const token = request.header('Authorization').split(' ')[1];
154     const kioskToken = await KioskToken.query()
155         .where({
156             Token
157         })
158         .first();
159     if (!kioskToken) {
160         throw new Error('Token is invalid!');
161     }
162     jwt.verify(token, Env.get('JWT_SECRET_KIOSK'));
163     const transaction = await Transaction.query()
164         .where({
165             Id
166         })
167         .first();
168     if (!transaction) {
169         throw new Error('Transaction not found!');
170     }
171     const checkTransactionPayemnt = await
transaction.transactionPayment().first();
172     if (checkTransactionPayemnt) {
173         throw new Error('Can not cancel transaction because transaction has
been paid!');
174     }
175     await transaction.delete();
176     return true;

```

```
177     }
178   },
179
180   Transaction: {
181     async account(_) {
182       const transaction = new Transaction();
183       transaction.newUp(_);
184       return transaction.account().first();
185     },
186     async kiosk(_) {
187       const transaction = new Transaction();
188       transaction.newUp(_);
189       return transaction.kiosk().first();
190     },
191     async payment(_) {
192       const transaction = new Transaction();
193       transaction.newUp(_);
194       return transaction.payment().first();
195     },
196     async member(_) {
197       return Member.query()
198         .listing()
199         .where({ member_id: _.member_id })
200         .first();
201     }
202   }
203 };
204
```

Source code Account.js

```
1  'use strict';
2
3  const jwt = require('jsonwebtoken');
4  const moment = require('moment');
5
6  const Database = use('Database');
7  const Env = use('Env');
8  const Hash = use('Hash');
9  const { sendEmailNotificationBalanceDeposit } = use('App/Helpers');
10 const Member = use('App/Models/Member');
11 const Account = use('App/Models/Account');
12 const BalanceHistory = use('App/Models/BalanceHistory');
13 const BalanceDeposit = use('App/Models/BalanceDeposit');
14 const KioskToken = use('App/Models/KioskToken');
15
16 module.exports = {
17   Query: {
18     async allAccounts(_, { offset, limit }, { auth }) {
19       await auth.check();
20       const query = Account.query();
21       if (offset) {
22         query.offset(offset);
23       }
24       if (limit) {
25         query.limit(limit);
26       }
27       return query;
28     },
29     async fetchAccount(_, { member_id }, { auth }) {
30       await auth.check();
31       const member = await Member.find(member_id);
32       if (!member) {
33         throw new Error('Member not found!');
34       }
35       const account = await member.account().first();
36       if (!account) {
37         return Account.create({ member_id, balance: 0, active: 0 });
38       }
39       return account;
40     }
41   },
42
43   Mutation: {
44     // RESET PIN
45     async resetPINAccount(_, { member_id, pin, pin_confirm }, { auth }) {
```

```

46     await auth.check();
47     if (pin !== pin_confirm) {
48         throw new Error('PIN confirmation not match!');
49     }
50     const member = await Member.findOrFail(member_id);
51     const account = await member.account().first();
52     if (!account) {
53         await Account.create({ member_id, balance: 0, active: 0 });
54     }
55     account.pin = await Hash.make(pin.toString());
56     await account.save();
57     return true;
58 },
59 // ADD BALANCE
60 async addBalanceAccount(_, { member_id, amount }, { auth }) {
61     await auth.check();
62     const member = await Member.findOrFail(member_id);
63     const account = await member
64         .account()
65         .active()
66         .first();
67     if (!account) {
68         throw new Error('Account is not active!');
69     }
70     const lastBalanceHistory = await BalanceHistory.query()
71         .listing()
72         .where({ member_id })
73         .orderBy('datetime', 'desc')
74         .first();
75     const newBalanceHistory = await BalanceHistory.create({
76         member_id,
77         activity_id: 1,
78         amount,
79         description: 'Pengisian saldo',
80         balance: lastBalanceHistory ? lastBalanceHistory.balance + amount :
amount
81     });
82     try {
83         const user = await auth.getUser();
84         await BalanceDeposit.create({
85             history_id: newBalanceHistory.id,
86             created_by: user.id
87         });
88         // Send Notification Email
89         if (member.member_email) {
90             sendEmailNotificationBalanceDeposit({ member, balanceHistory:
newBalanceHistory });

```

```

91     }
92     return true;
93   } catch (error) {
94     await newBalanceHistory.delete();
95     throw new Error('Failed to add balance!');
96   }
97 },
98 // ACTIVATE ACCOUNT
99 async activateAccount(_, { member_id, active }, { auth }) {
100   await auth.check();
101   const member = await Member.findOneOrFail(member_id);
102   const account = await member.account().first();
103   if (account) {
104     account.active = active;
105     await account.save();
106     return true;
107   }
108   await Account.create({ member_id, active });
109   return true;
110 },
111 // KIOSK UPDATE PIN
112 async kioskUpdatePINAccount(_, { member_id, pin_old, pin_new,
pin_new_confirm }, { request }) {
113   if (!request.header('Authorization')) {
114     throw new Error('Token is invalid!');
115   }
116   const token = request.header('Authorization').split(' ')[1];
117   const kioskToken = await KioskToken.query()
118     .where({
119       Token
120     })
121     .first();
122   if (!kioskToken) {
123     throw new Error('Token is invalid!');
124   }
125   jwt.verify(token, Env.get('JWT_SECRET_KIOSK'));
126
127   const member = await Member.findOneOrFail(member_id);
128   const account = await member.account().first();
129   if (!account) {
130     await Account.create({ member_id, balance: 0, active: 0 });
131   }
132   const pinVerified = await Hash.verify(pin_old.toString(),
account.pin);
133   if (!pinVerified) {
134     throw new Error('PIN Lama salah!');
135   }

```

```

136
137     if (pin_new !== pin_new_confirm) {
138         throw new Error('New PIN confirmation not match!');
139     }
140
141     account.pin = await Hash.make(pin_new.toString());
142     await account.save();
143     return true;
144 }
145 },
146 Account: {
147     async balanceDeposits(_, { offset, limit, startDate, endDate, orderBy
148 }) {
149         const account = new Account();
150         account.newUp(_);
151         const query = Database.select(
152             'account_balance_deposits.*',
153             'account_balance_histories.member_id',
154             'account_balance_histories.created_at as datetime'
155         )
156         .from('account_balance_deposits')
157         .leftJoin(
158             'account_balance_histories',
159             'account_balance_deposits.history_id',
160             'account_balance_histories.id'
161         )
162         .where('account_balance_histories.member_id', account.member_id);
163         if (startDate) {
164             const momentStartDate = moment(startDate);
165             if (!momentStartDate.isValid()) {
166                 throw new Error('startDate is not valid datetime!');
167             }
168             query.where('created_at', '>=', momentStartDate.format('YYYY-MM-
169 DD'));
170         }
171         if (endDate) {
172             const momentEndDate = moment(endDate).add(1, 'days');
173             if (!momentEndDate.isValid()) {
174                 throw new Error('endDate is not valid datetime!');
175             }
176             query.where('created_at', '<=', momentEndDate.format('YYYY-MM-
177 DD'));
178         }
179         if (offset) {
180             query.offset(offset);
181         }
182         if (limit) {
183             query.limit(limit);
184         }
185     }
186 }

```

```

180     query.limit(limit);
181   }
182   if (orderBy) {
183     query.orderBy(orderBy.split(',')[0], orderBy.split(',')[1]);
184   }
185   return query;
186 },
187 async countBalanceDeposits(_, { startDate, endDate }) {
188   const account = new Account();
189   account.newUp(_);
190   const query = Database.select('member_id')
191     .from('account_balance_deposits')
192     .leftJoin(
193       'account_balance_histories',
194       'account_balance_deposits.history_id',
195       'account_balance_histories.id'
196     )
197     .where('member_id', account.member_id);
198   if (startDate) {
199     const momentStartDate = moment(startDate);
200     if (!momentStartDate.isValid()) {
201       throw new Error('startDate is not valid datetime!');
202     }
203     query.where('created_at', '>=', momentStartDate.format('YYYY-MM-
204 DD'));
205   }
206   if (endDate) {
207     const momentEndDate = moment(endDate).add(1, 'days');
208     if (!momentEndDate.isValid()) {
209       throw new Error('endDate is not valid datetime!');
210     }
211     query.where('created_at', '<=', momentEndDate.format('YYYY-MM-
212 DD'));
213   }
214   const count = await query.count('member_id as total');
215   return count[0].total;
216 },
217 async balanceHistories(_, { offset, limit, startDate, endDate, orderBy
218 }) {
219   const account = new Account();
220   account.newUp(_);
221   const query = account
222     .balanceHistory()
223     .listing()
224     .where({ member_id: account.member_id });
225   if (startDate) {
226     const momentStartDate = moment(startDate);

```

```

224     if (!momentStartDate.isValid()) {
225         throw new Error('startDate is not valid datetime!');
226     }
227     query.where('created_at', '>=', momentStartDate.format('YYYY-MM-
DD'));
228 }
229 if (endDate) {
230     const momentEndDate = moment(endDate).add(1, 'days');
231     if (!momentEndDate.isValid()) {
232         throw new Error('endDate is not valid datetime!');
233     }
234     query.where('created_at', '<=', momentEndDate.format('YYYY-MM-
DD'));
235 }
236 if (offset) {
237     query.offset(offset);
238 }
239 if (limit) {
240     query.limit(limit);
241 }
242 if (orderBy) {
243     query.orderBy(orderBy.split(',')[0], orderBy.split(',')[1]);
244 }
245 return query;
246 },
247 async countBalanceHistories(_, { startDate, endDate }) {
248     const account = new Account();
249     account.newUp(_);
250     const query = account.balanceHistory();
251     if (startDate) {
252         const momentStartDate = moment(startDate);
253         if (!momentStartDate.isValid()) {
254             throw new Error('startDate is not valid datetime!');
255         }
256         query.where('created_at', '>=', momentStartDate.format('YYYY-MM-
DD'));
257     }
258     if (endDate) {
259         const momentEndDate = moment(endDate).add(1, 'days');
260         if (!momentEndDate.isValid()) {
261             throw new Error('endDate is not valid datetime!');
262         }
263         query.where('created_at', '<=', momentEndDate.format('YYYY-MM-
DD'));
264     }
265     const count = await query.count();
266     return count[0]['count(*)'];

```



```

267     },
268     async transactions(_, { offset, limit, startDate, endDate, orderBy }) {
269         const account = new Account();
270         account.newUp(_);
271         const query = account
272             .transactions()
273             .listing()
274             .has('transactionPayment')
275             .where({ member_id: account.member_id });
276         if (startDate) {
277             const momentStartDate = moment(startDate);
278             if (!momentStartDate.isValid()) {
279                 throw new Error('startDate is not valid datetime!');
280             }
281             query.where('created_at', '>=', momentStartDate.format('YYYY-MM-
DD'));
282         }
283         if (endDate) {
284             const momentEndDate = moment(endDate).add(1, 'days');
285             if (!momentEndDate.isValid()) {
286                 throw new Error('endDate is not valid datetime!');
287             }
288             query.where('created_at', '<=', momentEndDate.format('YYYY-MM-
DD'));
289         }
290         if (offset) {
291             query.offset(offset);
292         }
293         if (limit) {
294             query.limit(limit);
295         }
296         if (orderBy) {
297             query.orderBy(orderBy.split(',')[0], orderBy.split(',')[1]);
298         }
299         return query;
300     },
301     async countTransactions(_, { startDate, endDate }) {
302         const account = new Account();
303         account.newUp(_);
304         const query = account.transactions().has('transactionPayment');
305         if (startDate) {
306             const momentStartDate = moment(startDate);
307             if (!momentStartDate.isValid()) {
308                 throw new Error('startDate is not valid datetime!');
309             }
310             query.where('created_at', '>=', momentStartDate.format('YYYY-MM-
DD'));

```

```
311     }
312     if (endDate) {
313         const momentEndDate = moment(endDate).add(1, 'days');
314         if (!momentEndDate.isValid()) {
315             throw new Error('endDate is not valid datetime!');
316         }
317         query.where('created_at', '<=', momentEndDate.format('YYYY-MM-
DD'));
318     }
319     const count = await query.count();
320     return count[0]['count(*)'];
321 },
322 async member(_) {
323     const account = new Account();
324     account.newUp(_);
325     return account.member().first();
326 },
327 async balance(_) {
328     const account = new Account();
329     account.newUp(_);
330     const balanceHistory = await account
331         .balanceHistory()
332         .orderBy('created_at', 'desc')
333         .first();
334     return balanceHistory ? balanceHistory.balance : 0;
335 }
336 }
337 };
338
```

Source code Results.vue

```
1 <template>
2   <div class="has-navbar-main">
3     <nav class="fixed-top my-breadcrumb" aria-label="breadcrumb">
4       <ol class="breadcrumb">
5         <li class="breadcrumb-item">
6           <router-link class="navbar-item btn" to="/">
7             <span class="icon is-small"> <fa icon="home" /> </span>
8             </router-link>
9           </li>
10          <li class="breadcrumb-item active">
11            <span class="btn">Hasil Pencarian: "{{ searchQuery }}"</span>
12          </li>
13        </ol>
14      </nav>
15      <section class="section has-navbar-main-2">
16        <Pagination
17          :current="pagination.current"
18          :total="results.total"
19          :itemsPerPage="pagination.perPage"
20          :step="pagination.step"
21          :onChange="onPageChange"
22        />
23        <div class="container">
24          <br />
25          <EprintList :results="results.data" />
26          <br />
27        </div>
28      </section>
29      <ProgressCircle
30        v-show="isLoading"
31        maskMarginTop="128px"
32        maskBackgroundColor="rgba(255, 255, 255, 0)"
33      />
34    </div>
35 </template>
36
37 <script>
38 import { ipcRenderer } from 'electron';
39 import ProgressCircle from '@/components/modals/ProgressCircle.vue';
40 import EprintList from '@/components/elements/EprintList.vue';
41 import Pagination from './Pagination.vue';
42 export default {
43   name: 'Results',
44   components: { EprintList, Pagination, ProgressCircle },
```

```

45     data() {
46         return {
47             isLoading: false,
48             searchQuery: this.$route.query.search,
49             results: {},
50             pagination: {
51                 current: 1,
52                 perPage: 20,
53                 step: 2
54             }
55         };
56     },
57     methods: {
58         getResults() {
59             this.isLoading = true;
60             ipcRenderer.send('digilib.search.r-get-results', {
61                 q: this.searchQuery,
62                 page: this.pagination.current
63             });
64         },
65         onPageChange(page) {
66             this.pagination.current = page;
67         }
68     },
69     mounted() {
70         this.getResults();
71         ipcRenderer.on('digilib.search.m-send-results', (event, result) => {
72             this.results = result;
73             this.isLoading = false;
74         });
75         ipcRenderer.on('digilib.search.m-send-results-error', (event, error) =>
76         {
77             console.log(error);
78             this.isLoading = false;
79         });
80     },
81     watch: {
82         pagination: {
83             handler: 'getResults',
84             deep: true,
85             immediate: true
86         }
87     };
88 </script>
89 <style scoped></style>

```

Source code digilib.js

```
1  import axios from 'axios';
2  import cachios from 'cachios';
3  import fs from 'fs';
4  import os from 'os';
5  import path from 'path';
6  import { URL } from 'url';
7  import { KIOSK_TOKEN, EPRINTS_API } from './constants';
8
9  export default {
10   browseByYear: {
11     getAllYears: () =>
12       cachios.post(
13         `${EPRINTS_API}/years`,
14         {},
15         {
16           ttl: 86400
17         }
18       ),
19     getResults: (year, grouping) =>
20       cachios.post(
21         `${EPRINTS_API}/years/${year}`,
22         { grouping },
23         {
24           ttl: 3600
25         }
26       )
27   },
28   browseByCreator: {
29     getCreatorsIndexes: () =>
30       cachios.post(
31         `${EPRINTS_API}/creators/indexes`,
32         {},
33         {
34           ttl: 86400
35         }
36       ),
37     getCreatorsByIndex: index =>
38       cachios.post(
39         `${EPRINTS_API}/creators/indexed/${index}`,
40         {},
41         {
42           ttl: 86400
43         }
44       ),
45     getResults: creatorUId =>
```

```

46     cachios.post(
47       `${EPRINTS_API}/creators/${creatorUid}`,
48       {},
49       {
50         ttl: 86400
51       }
52     )
53   },
54   browseBySubject: {
55     getSubjects: () =>
56       cachios.post(
57         `${EPRINTS_API}/subjects`,
58         {},
59         {
60           ttl: 86400
61         }
62       ),
63     getResults: (subject, grouping) =>
64       cachios.post(
65         `${EPRINTS_API}/subjects/${subject}`,
66         { grouping },
67         {
68           ttl: 3600
69         }
70       )
71   },
72   browseByDivision: {
73     getDivisions: () =>
74       cachios.post(
75         `${EPRINTS_API}/divisions`,
76         {},
77         {
78           ttl: 86400
79         }
80       ),
81     getYears: division =>
82       cachios.post(
83         `${EPRINTS_API}/divisions/${division}`,
84         {},
85         {
86           ttl: 3600
87         }
88       ),
89     getResults: (division, year, grouping) =>
90       cachios.post(
91         `${EPRINTS_API}/divisions/${division}/${year}`,
92         { grouping },

```

```

93     {
94         ttl: 3600
95     }
96 )
97 },
98 search: {
99     getResults: (q, page) =>
100     cachios.post(
101         `${EPRINTS_API}/search`,
102         {
103             q,
104             page
105         },
106         {
107             ttl: 86400
108         }
109     )
110 },
111 eprint: {
112     getEprint: id =>
113     cachios.post(
114         `${EPRINTS_API}/eprint/${id}`,
115         {},
116         {
117             ttl: 86400
118         }
119     ),
120     downloadDoc: async (id, docid, filename) => {
121         const response = await axios({
122             method: 'GET',
123             url: `${EPRINTS_API}/eprint/document/${id}/${docid}/${filename}`,
124             headers: { Authorization: `Bearer ${KIOSK_TOKEN}` },
125             responseType: 'stream'
126         });
127         const downloadPath = path.resolve(os.tmpdir(),
128             'vol_tmp_view_doc.pdf');
129         response.data.pipe(fs.createWriteStream(downloadPath));
130         return new Promise((resolve, reject) => {
131             response.data.on('end', () => {
132                 resolve(new URL(`file:///${downloadPath}`).href);
133             });
134             response.data.on('error', () => {
135                 reject();
136             });
137         });
138     }

```

```
139 };  
140
```

Source code index.js

```
1  import { app, BrowserWindow, ipcMain } from 'electron'  
2  import utils from './utils';  
3  import digilib from './digilib';  
4  import userAuth from './userAuth';  
5  import user from './user';  
6  import trx from './trx';  
7  
8  if (process.env.NODE_ENV !== 'development') {  
9    global.__static = require('path').join(__dirname,  
10     '/static').replace(/\\/g, '\\\\') // eslint-disable-line  
11 }  
12  
13 let mainWindow;  
14 const winURL =  
15   process.env.NODE_ENV === 'development'  
16     ? 'http://localhost:9080'  
17     : `file://${__dirname}/index.html`;  
18  
19 function createWindow() {  
20   /**  
21    Initial window options  
22   */  
23   mainWindow = new BrowserWindow({  
24     kiosk: true,  
25     useContentSize: true,  
26     webPreferences: {  
27       nodeIntegration: true  
28     }  
29   });  
30   mainWindow.setMenu(null);  
31   mainWindow.loadURL(winURL);  
32  
33   mainWindow.on('closed', () => {  
34     mainWindow = null;  
35   });  
36 }  
37  
38 app.on('ready', createWindow);  
39
```



```

40 app.on('window-all-closed', () => {
41   if (process.platform !== 'darwin') {
42     app.quit();
43   }
44 });
45
46 app.on('activate', () => {
47   if (mainWindow === null) {
48     createWindow();
49   }
50 });
51
52 /*
53 IPC MAIN
54 */
55
56 /* DIGILIB */
57 // Browse by Year
58 ipcMain.on('digilib.browse-by-year.r-get-all-years', event => {
59   digilib.browseByYear
60     .getAllYears()
61     .then(response => {
62       event.sender.send('digilib.browse-by-year.m-send-all-years',
63         response.data);
64     })
65     .catch(error => {
66       event.sender.send('digilib.browse-by-year.m-send-all-years-error',
67         error);
68     });
69 });
70
71 ipcMain.on('digilib.browse-by-year.r-get-results', (event, args) => {
72   digilib.browseByYear
73     .getResults(args.year, args.grouping)
74     .then(response => {
75       event.sender.send('digilib.browse-by-year.m-send-results',
76         response.data.data);
77     })
78     .catch(error => {
79       event.sender.send('digilib.browse-by-year.m-send-results-error',
80         error);
81     });
82 });
83
84 // Browse by Creator
85 ipcMain.on('digilib.browse-by-creator.r-get-creators-indexes', event => {
86   digilib.browseByCreator

```

```

83     .getCreatorsIndexes()
84     .then(response => {
85         event.sender.send('digilib.browse-by-creator.m-send-creators-
indexes', response.data);
86     })
87     .catch(error => {
88         event.sender.send('digilib.browse-by-creator.m-send-creators-indexes-
error', error);
89     });
90 });
91
92 ipcMain.on('digilib.browse-by-creator.r-get-creators-by-index', (event,
args) => {
93     digilib.browseByCreator
94     .getCreatorsByIndex(args.index)
95     .then(response => {
96         event.sender.send('digilib.browse-by-creator.m-send-creators-by-
index', response.data);
97     })
98     .catch(error => {
99         event.sender.send('digilib.browse-by-creator.m-send-creators-by-
index-error', error);
100     });
101 });
102
103 ipcMain.on('digilib.browse-by-creator.r-get-results', (event, args) => {
104     digilib.browseByCreator
105     .getResults(args.creatorUid)
106     .then(response => {
107         event.sender.send('digilib.browse-by-creator.m-send-results',
response.data.data);
108     })
109     .catch(error => {
110         event.sender.send('digilib.browse-by-creator.m-send-results-error',
error);
111     });
112 });
113
114 // Browse by Subject
115 ipcMain.on('digilib.browse-by-subject.r-get-subject-list', event => {
116     digilib.browseBySubject
117     .getSubjects()
118     .then(response => {
119         event.sender.send('digilib.browse-by-subject.m-send-subject-list',
response.data);
120     })
121     .catch(error => {

```

```

122     event.sender.send('digilib.browse-by-subject.m-send-subject-list-
error', error);
123   });
124 });
125
126 ipcMain.on('digilib.browse-by-subject.r-get-results', (event, args) => {
127   digilib.browseBySubject
128     .getResults(args.subjectUid, args.grouping)
129     .then(response => {
130       event.sender.send('digilib.browse-by-subject.m-send-results',
response.data.data);
131     })
132     .catch(error => {
133       event.sender.send('digilib.browse-by-subject.m-send-results-error',
error);
134     });
135 });
136
137 // Browse by Division
138 ipcMain.on('digilib.browse-by-division.r-get-division-list', event => {
139   digilib.browseByDivision
140     .getDivisions()
141     .then(response => {
142       event.sender.send('digilib.browse-by-division.m-send-division-list',
response.data);
143     })
144     .catch(error => {
145       event.sender.send('digilib.browse-by-division.m-send-division-list-
error', error);
146     });
147 });
148
149 ipcMain.on('digilib.browse-by-division.r-get-years', (event, args) => {
150   digilib.browseByDivision
151     .getYears(args.division)
152     .then(response => {
153       event.sender.send('digilib.browse-by-division.m-send-years',
response.data);
154     })
155     .catch(error => {
156       event.sender.send('digilib.browse-by-division.m-send-years-error',
error);
157     });
158 });
159
160 ipcMain.on('digilib.browse-by-division.r-get-results', (event, args) => {
161   digilib.browseByDivision

```

```

162     .getResults(args.division, args.year, args.grouping)
163     .then(response => {
164         event.sender.send('digilib.browse-by-division.m-send-results',
165             response.data.data);
166     })
167     .catch(error => {
168         event.sender.send('digilib.browse-by-division.m-send-results-error',
169             error);
170     });
171 // Search
172 ipcMain.on('digilib.search.r-get-results', (event, args) => {
173     digilib.search
174     .getResults(args.q, args.page)
175     .then(response => {
176         event.sender.send('digilib.search.m-send-results', response.data);
177     })
178     .catch(error => {
179         event.sender.send('digilib.search.m-send-results-error', error);
180     });
181 });
182
183 // EPrint
184 ipcMain.on('digilib.eprint.r-get-eprint', (event, args) => {
185     digilib.eprint
186     .getEprint(args.id)
187     .then(response => {
188         event.sender.send('digilib.eprint.m-send-eprint', response.data);
189     })
190     .catch(error => {
191         event.sender.send('digilib.eprint.m-send-eprint-error', error);
192     });
193 });
194
195 ipcMain.on('digilib.eprint.r-download-doc', (event, args) => {
196     digilib.eprint
197     .downloadDoc(args.id, args.docid, args.filename)
198     .then(response => {
199         event.sender.send('digilib.eprint.m-send-download-doc', response);
200     })
201     .catch(error => {
202         event.sender.send('digilib.eprint.m-send-download-doc-error', error);
203     });
204 });
205
206 /* TRX */

```

```
207
208 ipcMain.on('trx.r-create-trx', (event, args) => {
209   console.log('creating trx');
210   trx
211     .createTrx(args)
212     .then(result => {
213       event.sender.send('trx.m-send-trx-created', result.data.transaction);
214     })
215     .catch(error => {
216       event.sender.send('trx.m-send-trx-error', error);
217     });
218 });
219
220 ipcMain.on('trx.r-cancel-trx', (event, args) => {
221   console.log('cancelling trx');
222   trx
223     .cancelTrx(args)
224     .then(result => {
225       event.sender.send('trx.m-send-trx-cancelled',
226         result.data.cancelTransaction);
227     })
228     .catch(error => {
229       event.sender.send('trx.m-send-error-cancel-trx', error);
230     });
231 });
232 ipcMain.on('trx.r-verify-trx-payment', (event, args) => {
233   console.log('verifying trx payment');
234   trx
235     .verifyTrxPayment(args)
236     .then(result => {
237       event.sender.send('trx.m-send-trx-payment-verified',
238         result.data.verified);
239     })
240     .catch(error => {
241       console.log(error);
242       event.sender.send('trx.m-send-trx-payment-error', error);
243     });
244 });
245 ipcMain.on('trx.r-send-print-document', (event, args) => {
246   console.log('printing document');
247   try {
248     event.sender.send('trx.m-send-print-document');
249     utils.printDocument(args);
250   } catch (e) {
251     event.sender.send('trx.m-send-print-document-error');
```

```

252   }
253 });
254
255 /* User Auth */
256
257 ipcMain.on('user-auth.r-send-fetch-member', (event, args) => {
258   console.log('fetch member');
259   userAuth
260     .fetchMember(args)
261     .then(result => {
262       event.sender.send('user-auth.m-send-member', result.data.member);
263     })
264     .catch(error => {
265       console.log(error);
266       event.sender.send('user-auth.m-send-member-error', error);
267     });
268 });
269
270 ipcMain.on('user-auth.r-verify-member-pin', (event, args) => {
271   console.log('verify member pin');
272   userAuth
273     .verifyMemberPin(args)
274     .then(result => {
275       event.sender.send('user-auth.m-member-pin-verified',
276         result.data.member);
277     })
278     .catch(error => {
279       console.log(error);
280       event.sender.send('user-auth.m-member-pin-error', error);
281     });
282 });
283 /* Update PIN */
284
285 ipcMain.on('user.r-send-update-pin', (event, args) => {
286   console.log('update pin');
287   user
288     .updatePin(args)
289     .then(result => {
290       event.sender.send('user.m-send-pin-updated', result.data);
291     })
292     .catch(error => {
293       console.log(error);
294       event.sender.send('user.m-send-update-pin-error', error);
295     });
296 });

```

Foto pengujian oleh pemustaka

