Source code AuthController.js

```
'use strict';

const Drive = use('Drive');
const User = use('App/Models/User');

class AuthController {
  async login({ request, auth }) {
    const { username, password } = request.all();
    const { token } = await auth.attempt(username, password);
    const user = await User.query()
      .profile()
      .where({
        username
      })
      .first();
    try {
      user.avatar = Buffer.from(await
Drive.get(`avatars/${user.avatar}`)).toString('base64');
    } catch (error) {
      user.avatar = Buffer.from(await
Drive.get('avatars/default.jpg')).toString('base64');
    }
    return {
      user,
      token
    };
  }
}

module.exports = AuthController;

```

Source code GraphQLController.js

```
'use strict';

const Config = use('Config');

const { graphqlAdonis } = require('apollo-server-adonis');
const { makeExecutableSchema } = require('graphql-tools');
const { fileLoader, mergeTypes, mergeResolvers } = require('merge-graphql-schemas');

class GraphQLController {
  constructor() {
    const typeDefs = mergeTypes(
      fileLoader(Config.get('graphql.schema'), {
        recursive: true
      })
    );
    const resolvers = mergeResolvers(fileLoader(Config.get('graphql.resolvers')));

    this.$schema = makeExecutableSchema({
      typeDefs,
      resolvers
    });
  }

  server(context) {
    return graphqlAdonis({
      context,
      schema: this.$schema,
      debug: false
    })(context);
  }
}

module.exports = GraphQLController;

```

Source code User.js

```
'use strict';

const Hash = use('Hash');
const User = use('App/Models/User');

module.exports = {
  Query: {
    async allUsers(_, { offset, limit, search }, { auth }) {
      await auth.check();
      const query = User.query();
      if (search) {
        query
          .where('name', 'like', `%${search}%`)
          .orWhere('username', 'like', `%${search}%`)
          .orWhere('email', 'like', `%${search}%`);
      }
      if (offset) {
        query.offset(offset);
      }
      if (limit) {
        query.limit(limit);
      }
      return query.orderBy('name', 'asc');
    },
    async countAllUsers(_, { search }, { auth }) {
      await auth.check();
      const query = User.query();
      if (search) {
        query
          .where('name', 'like', `%${search}%`)
          .orWhere('username', 'like', `%${search}%`)
          .orWhere('email', 'like', `%${search}%`);
      }
      const count = await query.count();
      return count[0]['count(*)'];
    },
    async fetchUser(_, { id }, { auth }) {
      await auth.check();
      return User.find(id);
    }
  },

  Mutation: {
    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

```
'use strict';

const jwt = require('jsonwebtoken');
const moment = require('moment');

const Env = use('Env');
const { formatPages, countPages } = use('App/Helpers');
const Account = use('App/Models/Account');
const Member = use('App/Models/Member');
const MemberTypePrice = use('App/Models/MemberTypePrice');
const Transaction = use('App/Models/Transaction');
const KioskToken = use('App/Models/KioskToken');

module.exports = {
  Query: {
    async allTransactions(_, { startDate, endDate, offset, limit, orderBy }, { auth }) {
      await auth.check();
      const query = Transaction.query()
        .listing()
        .has('transactionPayment');
      if (startDate) {
        const momentStartDate = moment(startDate);
        if (!momentStartDate.isValid()) {
          throw new Error('startDate is not valid datetime!');
        }
        query.where('created_at', '>=', momentStartDate.format('YYYY-MM-DD'));
      }
      if (endDate) {
        const momentEndDate = moment(endDate).add(1, 'days');
        if (!momentEndDate.isValid()) {
          throw new Error('endDate is not valid datetime!');
        }
        query.where('created_at', '<=', momentEndDate.format('YYYY-MM-DD'));
      }
      if (offset) {
        query.offset(offset);
      }
      if (limit) {
        query.limit(limit);
      }
      if (orderBy) {
        query.orderBy(orderBy.split(',')[0], orderBy.split(',')[1]);
```

```
      }
      return query;
    },
    async fetchTransaction(_, { id }, { auth }) {
      await auth.check();
      const transaction = await Transaction.query()
        .listing()
        .has('transactionPayment')
        .where({
          id
        })
        .first();
      if (!transaction) {
        throw new Error('Transaction not found!');
      }
      return transaction;
    },
    async countAllTransactions(_, { startDate, endDate }, { auth }) {
      await auth.check();
      const query = Transaction.query().has('transactionPayment');
      if (startDate) {
        const momentStartDate = moment(startDate);
        if (!momentStartDate.isValid()) {
          throw new Error('startDate is not valid datetime!');
        }
        query.where('created_at', '>=', momentStartDate.format('YYYY-MM-
DD'));
      }
      if (endDate) {
        const momentEndDate = moment(endDate).add(1, 'days');
        if (!momentEndDate.isValid()) {
          throw new Error('endDate is not valid datetime!');
        }
        query.where('created_at', '<=', momentEndDate.format('YYYY-MM-
DD'));
      }
      const count = await query.count();
      return count[0]['count(*)'];
    }
  },

  Mutation: {
    async createTransaction(_, { member_id, eprint_id, filename, pages }, {
request }) {
      const token = request.header('Authorization').split(' ')[1];
      const kioskToken = await KioskToken.query()
        .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.');
119       }
120       const { price } = memberTypeprice;
121       const total_price = num_pages * price;
122       const lastBalanceHistory = await account
123         .balanceHistory()
124         .listing()
125         .where({ member_id })
126         .orderBy('datetime', 'desc')
127         .first();
128
129       const balance = lastBalanceHistory ? lastBalanceHistory.balance : 0;
130
131       if (balance < total_price) {
```

```
        throw new Error('Saldo tidak mencukupi!');
      }

      const transaction = await Transaction.create({
        member_id,
        kiosk_id: decodedToken.uid,
        eprint_id,
        filename,
        pages: formattedPages,
        num_pages,
        price,
        total_price
      });
      return Transaction.query()
        .listing()
        .where({
          id: transaction.id
        })
        .first();
    },
    async cancelTransaction(_, { id }, { request }) {
      const token = request.header('Authorization').split(' ')[1];
      const kioskToken = await KioskToken.query()
        .where({
          Token
        })
        .first();
      if (!kioskToken) {
        throw new Error('Token is invalid!');
      }
      jwt.verify(token, Env.get('JWT_SECRET_KIOSK'));
      const transaction = await Transaction.query()
        .where({
          Id
        })
        .first();
      if (!transaction) {
        throw new Error('Transaction not found!');
      }
      const checkTransactionPayemnt = await
transaction.transactionPayment().first();
      if (checkTransactionPayemnt) {
        throw new Error('Can not cancel transaction because transaction has
been paid!');
      }
      await transaction.delete();
      return true;
```

```
    }
  },

  Transaction: {
    async account(_) {
      const transaction = new Transaction();
      transaction.newUp(_);
      return transaction.account().first();
    },
    async kiosk(_) {
      const transaction = new Transaction();
      transaction.newUp(_);
      return transaction.kiosk().first();
    },
    async payment(_) {
      const transaction = new Transaction();
      transaction.newUp(_);
      return transaction.payment().first();
    },
    async member(_) {
      return Member.query()
        .listing()
        .where({ member_id: _.member_id })
        .first();
    }
  }
};

```

Source code Account.js

```
'use strict';

const jwt = require('jsonwebtoken');
const moment = require('moment');

const Database = use('Database');
const Env = use('Env');
const Hash = use('Hash');
const { sendEmailNotificationBalanceDeposit } = use('App/Helpers');
const Member = use('App/Models/Member');
const Account = use('App/Models/Account');
const BalanceHistory = use('App/Models/BalanceHistory');
const BalanceDeposit = use('App/Models/BalanceDeposit');
const KioskToken = use('App/Models/KioskToken');

module.exports = {
  Query: {
    async allAccounts(_, { offset, limit }, { auth }) {
      await auth.check();
      const query = Account.query();
      if (offset) {
        query.offset(offset);
      }
      if (limit) {
        query.limit(limit);
      }
      return query;
    },
    async fetchAccount(_, { member_id }, { auth }) {
      await auth.check();
      const member = await Member.find(member_id);
      if (!member) {
        throw new Error('Member not found!');
      }
      const account = await member.account().first();
      if (!account) {
        return Account.create({ member_id, balance: 0, active: 0 });
      }
      return account;
    }
  },

  Mutation: {
    // RESET PIN
    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 });
```

```
        }
        return true;
      } catch (error) {
        await newBalanceHistory.delete();
        throw new Error('Failed to add balance!');
      }
    },
    // ACTIVATE ACCOUNT
    async activateAccount(_, { member_id, active }, { auth }) {
      await auth.check();
      const member = await Member.findOrFail(member_id);
      const account = await member.account().first();
      if (account) {
        account.active = active;
        await account.save();
        return true;
      }
      await Account.create({ member_id, active });
      return true;
    },
    // KIOSK UPDATE PIN
    async kioskUpdatePINAccount(_, { member_id, pin_old, pin_new,
pin_new_confirm }, { request }) {
      if (!request.header('Authorization')) {
        throw new Error('Token is invalid!');
      }
      const token = request.header('Authorization').split(' ')[1];
      const kioskToken = await KioskToken.query()
        .where({
          Token
        })
        .first();
      if (!kioskToken) {
        throw new Error('Token is invalid!');
      }
      jwt.verify(token, Env.get('JWT_SECRET_KIOSK'));

      const member = await Member.findOrFail(member_id);
      const account = await member.account().first();
      if (!account) {
        await Account.create({ member_id, balance: 0, active: 0 });
      }
      const pinVerified = await Hash.verify(pin_old.toString(),
account.pin);
      if (!pinVerified) {
        throw new Error('PIN Lama salah!');
      }
```

```
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       const account = new Account();
149       account.newUp(_);
150       const query = Database.select(
151         'account_balance_deposits.*',
152         'account_balance_histories.member_id',
153         'account_balance_histories.created_at as datetime'
154       )
155         .from('account_balance_deposits')
156         .leftJoin(
157           'account_balance_histories',
158           'account_balance_deposits.history_id',
159           'account_balance_histories.id'
160         )
161         .where('account_balance_histories.member_id', account.member_id);
162       if (startDate) {
163         const momentStartDate = moment(startDate);
164         if (!momentStartDate.isValid()) {
165           throw new Error('startDate is not valid datetime!');
166         }
167         query.where('created_at', '>=', momentStartDate.format('YYYY-MM-
    DD'));
168       }
169       if (endDate) {
170         const momentEndDate = moment(endDate).add(1, 'days');
171         if (!momentEndDate.isValid()) {
172           throw new Error('endDate is not valid datetime!');
173         }
174         query.where('created_at', '<=', momentEndDate.format('YYYY-MM-
    DD'));
175       }
176       if (offset) {
177         query.offset(offset);
178       }
179       if (limit) {
```

```
        query.limit(limit);
      }
      if (orderBy) {
        query.orderBy(orderBy.split(',')[0], orderBy.split(',')[1]);
      }
      return query;
    },
    async countBalanceDeposits(_, { startDate, endDate }) {
      const account = new Account();
      account.newUp(_);
      const query = Database.select('member_id')
        .from('account_balance_deposits')
        .leftJoin(
          'account_balance_histories',
          'account_balance_deposits.history_id',
          'account_balance_histories.id'
        )
        .where('member_id', account.member_id);
      if (startDate) {
        const momentStartDate = moment(startDate);
        if (!momentStartDate.isValid()) {
          throw new Error('startDate is not valid datetime!');
        }
        query.where('created_at', '>=', momentStartDate.format('YYYY-MM-
DD'));
      }
      if (endDate) {
        const momentEndDate = moment(endDate).add(1, 'days');
        if (!momentEndDate.isValid()) {
          throw new Error('endDate is not valid datetime!');
        }
        query.where('created_at', '<=', momentEndDate.format('YYYY-MM-
DD'));
      }
      const count = await query.count('member_id as total');
      return count[0].total;
    },
    async balanceHistories(_, { offset, limit, startDate, endDate, orderBy
}) {
      const account = new Account();
      account.newUp(_);
      const query = account
        .balanceHistory()
        .listing()
        .where({ member_id: account.member_id });
      if (startDate) {
        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

```
<template>
  <div class="has-navbar-main">
    <nav class="fixed-top my-breadcrumb" aria-label="breadcrumb">
      <ol class="breadcrumb">
        <li class="breadcrumb-item">
          <router-link class="navbar-item btn" to="/">
            <span class="icon is-small"> <fa icon="home" /> </span>
<span>Beranda</span>
          </router-link>
        </li>
        <li class="breadcrumb-item active">
          <span class="btn">Hasil Pencarian: "{{ searchQuery }}"</span>
        </li>
      </ol>
    </nav>
    <section class="section has-navbar-main-2">
      <Pagination
        :current="pagination.current"
        :total="results.total"
        :itemsPerPage="pagination.perPage"
        :step="pagination.step"
        :onChange="onPageChange"
      />
      <div class="container">
        <br />
        <EprintList :results="results.data" />
        <br />
      </div>
    </section>
    <ProgressCircle
      v-show="isLoading"
      maskMarginTop="128px"
      maskBackgroundColor="rgba(255, 255, 255, 0)"
    />
  </div>
</template>

<script>
import { ipcRenderer } from 'electron';
import ProgressCircle from '@/components/modals/ProgressCircle.vue';
import EprintList from '@/components/elements/EprintList.vue';
import Pagination from './Pagination.vue';
export default {
  name: 'Results',
  components: { EprintList, Pagination, ProgressCircle },
```

```
  data() {
    return {
      isLoading: false,
      searchQuery: this.$route.query.search,
      results: {},
      pagination: {
        current: 1,
        perPage: 20,
        step: 2
      }
    };
  },
  methods: {
    getResults() {
      this.isLoading = true;
      ipcRenderer.send('digilib.search.r-get-results', {
        q: this.searchQuery,
        page: this.pagination.current
      });
    },
    onPageChange(page) {
      this.pagination.current = page;
    }
  },
  mounted() {
    this.getResults();
    ipcRenderer.on('digilib.search.m-send-results', (event, result) => {
      this.results = result;
      this.isLoading = false;
    });
    ipcRenderer.on('digilib.search.m-send-results-error', (event, error) =>
{
      console.log(error);
      this.isLoading = false;
    });
  },
  watch: {
    pagination: {
      handler: 'getResults',
      deep: true,
      immediate: true
    }
  }
};
</script>
<style scoped></style>
```

Source code digilib.js

```
import axios from 'axios';
import cachios from 'cachios';
import fs from 'fs';
import os from 'os';
import path from 'path';
import { URL } from 'url';
import { KIOSK_TOKEN, EPRINTS_API } from './constants';

export default {
  browseByYear: {
    getAllYears: () =>
      cachios.post(
        `${EPRINTS_API}/years`,
        {},
        {
          ttl: 86400
        }
      ),
    getResults: (year, grouping) =>
      cachios.post(
        `${EPRINTS_API}/years/${year}`,
        { grouping },
        {
          ttl: 3600
        }
      )
  },
  browseByCreator: {
    getCreatorsIndexes: () =>
      cachios.post(
        `${EPRINTS_API}/creators/indexes`,
        {},
        {
          ttl: 86400
        }
      ),
    getCreatorsByIndex: index =>
      cachios.post(
        `${EPRINTS_API}/creators/indexed/${index}`,
        {},
        {
          ttl: 86400
        }
      ),
    getResults: creatorUid =>
```

```
      cachios.post(
        `${EPRINTS_API}/creators/${creatorUid}`,
        {},
        {
          ttl: 86400
        }
      )
  },
  browseBySubject: {
    getSubjects: () =>
      cachios.post(
        `${EPRINTS_API}/subjects`,
        {},
        {
          ttl: 86400
        }
      ),
    getResults: (subject, grouping) =>
      cachios.post(
        `${EPRINTS_API}/subjects/${subject}`,
        { grouping },
        {
          ttl: 3600
        }
      )
  },
  browseByDivision: {
    getDivisions: () =>
      cachios.post(
        `${EPRINTS_API}/divisions`,
        {},
        {
          ttl: 86400
        }
      ),
    getYears: division =>
      cachios.post(
        `${EPRINTS_API}/divisions/${division}`,
        {},
        {
          ttl: 3600
        }
      ),
    getResults: (division, year, grouping) =>
      cachios.post(
        `${EPRINTS_API}/divisions/${division}/${year}`,
        { grouping },
```

```
        {
          ttl: 3600
        }
      )
  },
  search: {
    getResults: (q, page) =>
      cachios.post(
        `${EPRINTS_API}/search`,
        {
          q,
          page
        },
        {
          ttl: 86400
        }
      )
    },
  eprint: {
    getEprint: id =>
      cachios.post(
        `${EPRINTS_API}/eprint/${id}`,
        {},
        {
          ttl: 86400
        }
      ),
    downloadDoc: async (id, docid, filename) => {
      const response = await axios({
        method: 'GET',
        url: `${EPRINTS_API}/eprint/document/${id}/${docid}/${filename}`,
        headers: { Authorization: `Bearer ${KIOSK_TOKEN}` },
        responseType: 'stream'
      });
      const downloadPath = path.resolve(os.tmpdir(),
'vol_tmp_view_doc.pdf');
      response.data.pipe(fs.createWriteStream(downloadPath));
      return new Promise((resolve, reject) => {
        response.data.on('end', () => {
          resolve(new URL(`file:///${downloadPath}`).href);
        });
        response.data.on('error', () => {
          reject();
        });
      });
    }
  }
```

```
};

```

Source code index.js

```
import { app, BrowserWindow, ipcMain } from 'electron'
import utils from './utils';
import digilib from './digilib';
import userAuth from './userAuth';
import user from './user';
import trx from './trx';

if (process.env.NODE_ENV !== 'development') {
  global.__static = require('path').join(__dirname,
'/static').replace(/\\/g, '\\\\') // eslint-disable-line
}

let mainWindow;
const winURL =
  process.env.NODE_ENV === 'development'
    ? 'http://localhost:9080'
    : `file://${__dirname}/index.html`;

function createWindow() {
  /**
  Initial window options
  */
  mainWindow = new BrowserWindow({
    kiosk: true,
    useContentSize: true,
    webPreferences: {
      nodeIntegration: true
    }
  });

  mainWindow.setMenu(null);
  mainWindow.loadURL(winURL);

  mainWindow.on('closed', () => {
    mainWindow = null;
  });
}

app.on('ready', createWindow);

```

```
app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') {
    app.quit();
  }
});

app.on('activate', () => {
  if (mainWindow === null) {
    createWindow();
  }
});

/*
IPC MAIN
*/

/* DIGILILB */
// Browse by Year
ipcMain.on('digilib.browse-by-year.r-get-all-years', event => {
  digilib.browseByYear
    .getAllYears()
    .then(response => {
      event.sender.send('digilib.browse-by-year.m-send-all-years',
response.data);
    })
    .catch(error => {
      event.sender.send('digilib.browse-by-year.m-send-all-years-error',
error);
    });
});

ipcMain.on('digilib.browse-by-year.r-get-results', (event, args) => {
  digilib.browseByYear
    .getResults(args.year, args.grouping)
    .then(response => {
      event.sender.send('digilib.browse-by-year.m-send-results',
response.data.data);
    })
    .catch(error => {
      event.sender.send('digilib.browse-by-year.m-send-results-error',
error);
    });
});

// Browse by Creator
ipcMain.on('digilib.browse-by-creator.r-get-creators-indexes', event => {
  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 => {
```

```
      event.sender.send('digilib.browse-by-subject.m-send-subject-list-error', error);
    });
});

ipcMain.on('digilib.browse-by-subject.r-get-results', (event, args) => {
  digilib.browseBySubject
    .getResults(args.subjectUid, args.grouping)
    .then(response => {
      event.sender.send('digilib.browse-by-subject.m-send-results', response.data.data);
    })
    .catch(error => {
      event.sender.send('digilib.browse-by-subject.m-send-results-error', error);
    });
});

// Browse by Division
ipcMain.on('digilib.browse-by-division.r-get-division-list', event => {
  digilib.browseByDivision
    .getDivisions()
    .then(response => {
      event.sender.send('digilib.browse-by-division.m-send-division-list', response.data);
    })
    .catch(error => {
      event.sender.send('digilib.browse-by-division.m-send-division-list-error', error);
    });
});

ipcMain.on('digilib.browse-by-division.r-get-years', (event, args) => {
  digilib.browseByDivision
    .getYears(args.division)
    .then(response => {
      event.sender.send('digilib.browse-by-division.m-send-years', response.data);
    })
    .catch(error => {
      event.sender.send('digilib.browse-by-division.m-send-years-error', error);
    });
});

ipcMain.on('digilib.browse-by-division.r-get-results', (event, args) => {
  digilib.browseByDivision
```

```
    .getResults(args.division, args.year, args.grouping)
    .then(response => {
      event.sender.send('digilib.browse-by-division.m-send-results',
response.data.data);
    })
    .catch(error => {
      event.sender.send('digilib.browse-by-division.m-send-results-error',
error);
    });
});

// Search
ipcMain.on('digilib.search.r-get-results', (event, args) => {
  digilib.search
    .getResults(args.q, args.page)
    .then(response => {
      event.sender.send('digilib.search.m-send-results', response.data);
    })
    .catch(error => {
      event.sender.send('digilib.search.m-send-results-error', error);
    });
});

// EPrint
ipcMain.on('digilib.eprint.r-get-eprint', (event, args) => {
  digilib.eprint
    .getEprint(args.id)
    .then(response => {
      event.sender.send('digilib.eprint.m-send-eprint', response.data);
    })
    .catch(error => {
      event.sender.send('digilib.eprint.m-send-eprint-error', error);
    });
});

ipcMain.on('digilib.eprint.r-download-doc', (event, args) => {
  digilib.eprint
    .downloadDoc(args.id, args.docid, args.filename)
    .then(response => {
      event.sender.send('digilib.eprint.m-send-download-doc', response);
    })
    .catch(error => {
      event.sender.send('digilib.eprint.m-send-download-doc-error', error);
    });
});

/* 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',
    result.data.cancelTransaction);
226     })
227     .catch(error => {
228       event.sender.send('trx.m-send-error-cancel-trx', error);
229     });
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',
    result.data.verified);
238     })
239     .catch(error => {
240       console.log(error);
241       event.sender.send('trx.m-send-trx-payment-error', error);
242     });
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');
```

```
  }
});

/* User Auth */

ipcMain.on('user-auth.r-send-fetch-member', (event, args) => {
  console.log('fetch member');
  userAuth
    .fetchMember(args)
    .then(result => {
      event.sender.send('user-auth.m-send-member', result.data.member);
    })
    .catch(error => {
      console.log(error);
      event.sender.send('user-auth.m-send-member-error', error);
    });
});

ipcMain.on('user-auth.r-verify-member-pin', (event, args) => {
  console.log('verify member pin');
  userAuth
    .verifyMemberPin(args)
    .then(result => {
      event.sender.send('user-auth.m-member-pin-verified',
result.data.member);
    })
    .catch(error => {
      console.log(error);
      event.sender.send('user-auth.m-member-pin-error', error);
    });
});

/* Update PIN */

ipcMain.on('user.r-send-update-pin', (event, args) => {
  console.log('update pin');
  user
    .updatePin(args)
    .then(result => {
      event.sender.send('user.m-send-pin-updated', result.data);
    })
    .catch(error => {
      console.log(error);
      event.sender.send('user.m-send-update-pin-error', error);
    });
});
```

Foto pengujian oleh pemustaka