INTEROCO
www.interoco.com

INTEROCO COPYRIGHT OFFICE

Headquarters: INTEROCO - Copyright Office UG
Düsseldorfer Str. 39, Berlin, 10707, Federal Republic of Germany

Source Code of 'Travelpayouts'

Program Name:	**Source Code of 'Travelpayouts'**
Annotation (short description):	Travelpayouts functions as a multi-party CPA marketing platform that connects online resource owners (partners) with travel service providers (advertisers). Source code of Travelpayouts is executing following functions: • **Affiliate platform**: Travelpayouts acts as an intermediary between partners and advertisers • **Attracting clients:** Partners place special links on their resources leading to the websites of advertisers cooperating with the platform • **Remuneration:** For each attracted client who has performed the target action, the partner receives a commission from Travelpayouts • **Additional services:** The platform provides advertisers with various tools for tracking statistics, managing affiliate links and receiving payments Listing of source code is attached.
Computer Type:	Desktop, Mobile Devices, others
Programming languages:	Go, Typescript, Ruby, Javascript, Python, SQL, HTML, CSS, and others
Operating system:	Microsoft, Linux, iOS, Android, others

Listing of Source Code of 'Travelpayouts'

```
import './styles/app.scss';
import styles from './ui/layout.module.scss';

import { useEffect, useState } from 'react';
import { useNavigate } from 'react-router-dom';
import classNames from 'classnames';
import { observer } from 'mobx-react-lite';
import type { LocaleCode } from 'tp-design-system';
import { Loader } from 'tp-design-system';
import { useRollbarPerson } from '@rollbar/react';
import { Alert } from '@travelpayouts/alert';

import { errorHandler } from 'shared/api';
import { DEFAULT_ADMIN_PARTNER_MARKER } from 'shared/config';
import { formatSource } from 'shared/lib';
import { $locales, $rootStore, $sources, $user } from 'shared/model';

import { AppNotifications } from './ui/app_notifications/AppNotifications';
import { AppLayout } from './ui/AppLayout';

type States = 'idle' | 'loading' | 'failed' | 'userNotFound';

export const App = observer(() => {
  const navigate = useNavigate();
  const [loading, setLoading] = useState<States>('loading');
  const { language, loadLocale, setLocales } = $locales;
  const { init, setupApp } = $rootStore;

  useRollbarPerson({
    marker: $user.marker,
    email: $user.email
  });

  useEffect(() => {
    init(navigate);
    fetchData();
  }, []);

  useEffect(() => {
    // загружаем уведомления только для залогиненных пользователей
    if ($user.marker) {
      $user.getNotifications();
    }
  }, [language, $user.marker]);

  const fetchData = async () => {
    setLoading('loading');

    try {
      const response = await $user.getData();
```

```
      const locales = response.currentPartner.locales.map(
        locale => locale.toLowerCase() as LocaleCode
      );

      // в сторе локалей language == определившийся язык системы (или куки)
      // проверяем, что если в пришедших данных только 1 допустимый язык и он не совпадает с
автоопределённым,
      // то загружаем пришедшую локаль; иначе загружаем автоопределённую
      await loadLocale(locales.length === 1 && locales[0] !== language ? locales[0] : language);

      // сохраняем массив доступных локалей
      setLocales(locales);

      // остальные инит процедуры TODO: перевести на независимые сторы
      setupApp(response);

      // сетим данные по проектам
      response.currentPartner.sources?.forEach(source => $sources.add(formatSource(source)));

      setLoading('idle');
    } catch (error: unknown) {
      const customError = errorHandler(error);

      if (customError.code && ![205, 401].includes(customError.code)) {
        setLoading('failed');
      } else if (customError.code === 205) {
        setLoading('userNotFound');
      }
    }
  };

  if (loading === 'loading') {
    return (
      <div className={classNames('page-loading', styles.loader)} data-cy="app-loading">
        <Loader size="2xl" />
      </div>
    );
  }

  if (loading === 'failed') {
    return (
      <div className="app_failed">
        <Alert theme="error" data-cy="app-failed" className="app_failed-alert">
          {language === 'ru'
            ? 'Кажется, что-то пошло не так. Пожалуйста перезагрузите страницу.'
            : 'Oops, something went wrong. Please reload the page.'}
        </Alert>
      </div>
    );
  }

  if (loading === 'userNotFound') {
    const params = new URLSearchParams(window.location.search);

    if (!params.get('partner_marker')) {
      return null;
    }
```

```
    params.set('partner_marker', DEFAULT_ADMIN_PARTNER_MARKER);

    return (
      <div className="app_failed">
        <Alert theme="error" data-cy="app-failed" className="app_failed-alert">
          {language === 'ru' ? (
            <>
              Такого партнёра не существует.{' '}
              <a href={`${window.location.pathname}?${params.toString()}`}>
                Перейти на дефолтного 11501
              </a>
            </>
          ) : (
            <>
              This partner doesn’t exist.{' '}
              <a href={`${window.location.pathname}?${params.toString()}`}>
                Switch to default 11501
              </a>
            </>
          )}
        </Alert>
      </div>
    );
  }

  return (
    <>
      <AppLayout />
      <AppNotifications />
    </>
  );
});

import { Navigate, Route, Routes } from 'react-router-dom';
import { observer } from 'mobx-react-lite';

import { routesMap } from 'shared/config';
import { $rootStore, $user } from 'shared/model';
import type { RouteWithPermissionProps } from 'shared/types';

import { Layout } from './layout/Layout';
import { Academy } from './academy';
import { AllTools } from './all_tools';
import { Bonuses } from './bonuses';
import { BookingBooster } from './booking_booster';
import { Dashboard } from './dashboard';
import { Emerald } from './emerald';
import { Finance } from './finance';
import { LinkSwitcher } from './link_switcher';
import { Profile } from './profile';
import { ProgramView } from './program_view';
import { Programs, ProgramsConnected } from './programs';
import { Statistics } from './statistics';
import { WLApp } from './wl_app';

const routes: RouteWithPermissionProps[] = [
```

```
  {
    index: true,
    path: 'dashboard',
    element: <Dashboard />,
    permission: 'DASHBOARD.PARTNER_VIEW'
  },
  {
    path: 'emerald',
    element: <Emerald />,
    permission: 'EMERALD_SETTINGS.PARTNER_VIEW',
    featureFlag: 'emerald'
  },
  {
    path: 'emerald/:source_id',
    element: <Emerald />,
    permission: 'EMERALD_SETTINGS.PARTNER_VIEW',
    featureFlag: 'emerald'
  },
  {
    path: 'link_switcher/*',
    element: <LinkSwitcher />,
    permission: 'LINK_SWITCHER_SETTING.PARTNER_VIEW',
    featureFlag: 'link_switcher'
  },
  {
    path: 'wl_app',
    element: <WLApp />,
    permission: 'PAID_SERVICE.PARTNER_VIEW'
  },
  {
    path: 'all_tools',
    element: <AllTools />
  },
  {
    path: 'programs',
    element: <Programs />
  },
  {
    path: 'programs/connected',
    element: <ProgramsConnected />
  },
  {
    path: 'profile/*',
    element: <Profile />,
    permission: [
      'AD_LAW.PARTNER_VIEW',
      'PROFILE.PARTNER_VIEW',
      'SOURCE.PARTNER_EDIT',
      'SOURCE.PARTNER_VIEW'
    ]
  },
  {
    path: 'finance/*',
    element: <Finance />,
    permission: ['BALANCE.PARTNER_VIEW', 'PAYMENT.PARTNER_VIEW',
'AFFILIATE_REQUISITE.PARTNER_VIEW']
  },
```

```
  {
    path: 'statistics/*',
    element: <Statistics />,
    permission: ['PROGRAM_STATISTIC.PARTNER_VIEW', 'REFERRAL.PARTNER_VIEW']
  },
  {
    path: 'bonuses',
    element: <Bonuses />,
    permission: 'BONUSES.PARTNER_VIEW'
  },
  {
    path: 'academy',
    element: <Academy />
  },
  {
    path: 'programs/0/*',
    element: <Navigate replace={true} to={routesMap.statistics.referrals} />
  },
  {
    path: 'booking_booster',
    element: <BookingBooster />,
    permission: 'INTEGRATION_SETTINGS.PARTNER_VIEW',
    featureFlag: 'booking_booster'
  }
];

export const Router = observer(() => {
  const { featureType } = $rootStore;
  const { checkPermissions } = $user;

  const pages = routes
    .filter(({ permission }) => checkPermissions(permission))
    .filter(({ featureFlag }) => (featureFlag ? featureType(featureFlag) === 'B' : true));

  const defaultRoute = pages[0].path;

  return (
    <Routes>
      <Route path="/" element={<Layout />}>
        {pages.map(page => (
          <Route key={page.path} {...page} />
        ))}
        <Route path="/" element={<Navigate replace={true} to={{ pathname: defaultRoute }} />} />
      </Route>

      <Route path="programs/:program_id/*" element={<ProgramView />} />

      <Route path="*" element={<Navigate replace={true} to={{ pathname: defaultRoute }} />} />
    </Routes>
  );
});

@import 'shared/styles';
@import './external';
@import './overrides';

:root {
```

```
  --typography-link-color: var(--lk-text-link);
  --typography-link-hover-color: var(--lk-text-link-hover);
}

.page-loading {
  display: flex;
  flex: 1;
  align-items: center;
  justify-content: center;
  width: 100%;
  height: 100%;
  min-height: var(--spacing-1900);

  &.page-loading--global {
    position: relative;
    z-index: 15;
    background-color: var(--lk-bg);
  }

  .content & {
    height: 442px;
    min-height: auto !important;
  }

  &.layout_loading {
    position: fixed;
    top: 0;
    left: 0;
    z-index: 99999;
    width: 100%;
    height: 100%;
    background-color: rgb(255 255 255 / 0.15);

    [data-theme='dark'] & {
      background-color: rgb(41 43 55 / 0.15);
    }

    ~ .layout_scroll {
      filter: blur(2px);
    }
  }
}

.app_failed {
  display: flex;
  width: 100%;
  min-height: 100vh;
}

.app_failed-alert {
  margin: auto;

  a {
    text-decoration: underline;
  }
}
```

```
.screen-locker {
  overflow: hidden;

  body {
    overflow: hidden;
  }
}

import styles from './layout.module.scss';

import { lazy, Suspense, useEffect, useMemo } from 'react';
import { useLocation } from 'react-router-dom';
import classNames from 'classnames';
import { observer } from 'mobx-react-lite';
import { Loader } from 'tp-design-system';

import { CabinetPromo } from 'entities/cabinet_promo';
import { LsInstallModal } from 'entities/ls_activation';
import { Survey } from 'entities/survey';
import { Permission } from 'entities/user';
import { HeaderBalances } from 'features/header_balances';
import { IntroSteps } from 'features/intro_steps';
import { ActionModal, useActionDetailWatcher } from 'features/modal_action';
import { ReferralModal } from 'features/referral_modal';
import { SourcesWizard } from 'features/sources_wizard';
import { Router } from 'pages';
import { UTM_KEYS } from 'shared/config';
import { modalEmmiter, spEvent } from 'shared/lib';
import { $locales, $rootStore, $user } from 'shared/model';
import { ModalsEnum } from 'shared/types';

import { AppAlert } from './app_alert/AppAlert';
import { Footer } from './footer/Footer';
import { Header } from './header/Header';
import { Reactivation } from './modals/Reactivation';
import { Sidebar } from './sidebar/Sidebar';
import {
  useAppAnalytics,
  useInitSideApps,
  usePartnerMarkerWatcher,
  useRealVhUnits,
  useUserNotifications
} from '../lib';
import { useShowIntroSteps } from '../lib/useShowIntroSteps';

const AdminWidget = lazy(
  () => import(/* webpackChunkName: "adminWidget" */ 'entities/admin_widget')
);

const Banner = lazy(() => import(/* webpackChunkName: "cabinetPromo" */ './banner/Banner'));

modalEmmiter.registerModal(ModalsEnum.SourcesCreate, SourcesWizard.Create);
modalEmmiter.registerModal(ModalsEnum.SourcesManage, SourcesWizard.Manage);
modalEmmiter.registerModal(ModalsEnum.LsInstallModal, LsInstallModal);
modalEmmiter.registerModal(ModalsEnum.ActionModal, ActionModal);

export const AppLayout = observer(() => {
```

```
const { search } = useLocation();
const { layoutLoading, modalStore, surveyStore } = $rootStore;

useAppAnalytics();
useInitSideApps();
useRealVhUnits();
usePartnerMarkerWatcher();
useUserNotifications();
useActionDetailWatcher();

const showIntroSteps = useShowIntroSteps();

useEffect(() => {
  const searchParams = new URLSearchParams(search);
  const utmContent = searchParams.get('utm_content')?.toLowerCase();

  if (
    !showIntroSteps &&
    $locales.language === 'en' &&
    !surveyStore.getSurvey('reactivation') &&
    (utmContent === 'linkswitcher' || utmContent === 'invalidlinks')
  ) {
    modalEmmiter.registerModal(ModalsEnum.Reactivation, Reactivation);
    modalEmmiter.open(ModalsEnum.Reactivation);

    const utms: Record<string, string> = {};

    UTM_KEYS.forEach(key => {
      const value = searchParams.get(key);

      if (value) {
        utms[key] = value;
      }
    });

    spEvent({
      category: 'Dashboard',
      action: 'modal_opened',
      context: {
        type: 'ls_email_promo',
        info: utmContent === 'linkswitcher' ? 'ls_general' : 'invalid_links',
        user_data: utms
      }
    });
  }
}, [$locales.language, showIntroSteps]);

const showAiwp = useMemo(() => {
  return window.location.href.includes('whitepaper=aiwp') && !surveyStore.getSurvey('aiwp');
}, []);

if (showIntroSteps) {
  return <IntroSteps />;
}

return (
  <>
```

```
      {($locales.loading || layoutLoading) && (
        <div className={classNames('page-loading', styles.loader, styles.loaderLayout)}>
          <Loader size="2xl" />
        </div>
      )}

      <div className={styles.layout}>
        <Sidebar className={styles.sidebar} />

        <div className={styles.wrapper}>
          <AppAlert />

          <div className={styles.inner}>
            <div className={styles.container}>
              <div className={styles.header}>
                <Header />

                <div className={styles.balances} data-testid="balance">
                  <Permission permission="BALANCE.PARTNER_VIEW">
                    <HeaderBalances />
                  </Permission>
                </div>
              </div>

              <Router />

              <Footer className={styles.footer} />
            </div>
          </div>
        </div>
      </div>

      {modalStore.id === 'referral_modal' && <ReferralModal />}

      <Survey />

      <Suspense
        fallback={
          <div className={classNames('page-loading', styles.loader)}>
            <Loader size="2xl" />
          </div>
        }
      >
        <Permission permission="CABINET_PROMO.PARTNER_VIEW">
          <CabinetPromo />
        </Permission>

        {showAiwp && <Banner />}

        {$user.isAdmin && <AdminWidget />}
      </Suspense>
    </>
  );
});

import type { NavigateFunction } from 'react-router-dom';
import { makeAutoObservable } from 'mobx';
```

```
import type { AdLawStatus } from 'entities/adLaw';
import { AdLawStore } from 'entities/adLaw';
import { EmeraldStore } from 'entities/emerald';
import { SurveyStore } from 'entities/survey';
import { LinksGeneratorStore } from 'features/links_generator';
import { ModalActionStore } from 'features/modal_action';
import { AcademyStore } from 'pages/academy';
import { AllToolsStore } from 'pages/all_tools';
import { BonusesStore } from 'pages/bonuses';
import type { DashboardSettings } from 'pages/dashboard';
import { DashboardStore } from 'pages/dashboard';
import { EmeraldPageStore } from 'pages/emerald';
import { LinkSwitcherStore } from 'pages/link_switcher';
import { ProfileStore } from 'pages/profile';
import { ProgramViewStore } from 'pages/program_view';
import { ProgramsStore } from 'pages/programs';
import { StatisticsStore } from 'pages/statistics';
import type { CurrentPartnerQuery } from 'shared/api/sdk/partners.operations';

import { ModalStore } from '../ModalStore';

export class RootStore {
  programViewStore: ProgramViewStore;
  programsStore: ProgramsStore;
  modalStore: ModalStore;
  modalActionStore: ModalActionStore;
  dashboardStore: DashboardStore;
  statisticsStore: StatisticsStore;
  allToolsStore: AllToolsStore;
  academyStore: AcademyStore;
  bonusesStore: BonusesStore;
  linksGeneratorStore: LinksGeneratorStore;
  surveyStore: SurveyStore;
  profileStore: ProfileStore;
  adLaw: AdLawStore;

  emeraldStore: EmeraldStore;
  emeraldPageStore: EmeraldPageStore;
  linkSwitcherStore: LinkSwitcherStore;

  layoutLoading: boolean = false;

  navigate?: NavigateFunction;
  featureFlags = new Map<string, string>();

  constructor() {
    makeAutoObservable(this);

    this.programViewStore = new ProgramViewStore(this);
    this.programsStore = new ProgramsStore(this);
    this.modalStore = new ModalStore();
    this.modalActionStore = new ModalActionStore(this);
    this.dashboardStore = new DashboardStore(this);
    this.statisticsStore = new StatisticsStore(this);
    this.allToolsStore = new AllToolsStore();
    this.academyStore = new AcademyStore();
```

```
    this.bonusesStore = new BonusesStore(this);
    this.linksGeneratorStore = new LinksGeneratorStore(this);
    this.surveyStore = new SurveyStore(this);
    this.profileStore = new ProfileStore(this);
    this.adLaw = new AdLawStore();
    this.emeraldStore = new EmeraldStore();
    this.emeraldPageStore = new EmeraldPageStore(this);
    this.linkSwitcherStore = new LinkSwitcherStore(this);
  }

  init = async (navigate: NavigateFunction) => {
    this.navigate = navigate;
  };

  setupApp = ({
    currentPartner,
    flagrEvaluations,
    affiliateHints,
    authorizedUser
  }: CurrentPartnerQuery) => {
    flagrEvaluations.forEach(({ flagKey, variantKey }) => {
      this.featureFlags.set(flagKey, variantKey);
    });

    if (currentPartner.dashboardSettings) {
      this.dashboardStore.settings.set(currentPartner.dashboardSettings as DashboardSettings);
    }

    // если авторизован админ, то surveys берем из currentPartner + заполненные админские, чтобы админ
видел кабинет под партнёром корректно
    // если же авторизован сам юзер или его делегат, то берем surveys из authorizedUser т.к. для партнёра
и делегата свои наборы опросов
    const surveys =
      authorizedUser?.admin && authorizedUser?.surveys?.length && currentPartner?.surveys?.length
        ? [...authorizedUser.surveys, ...currentPartner.surveys]
        : authorizedUser?.surveys;

    this.surveyStore.init(surveys ?? [], affiliateHints ?? []);

    if (this.featureType('ad_law') === 'B') {
      const { adFormVisible, adLawInfo, adLawStatus } = currentPartner.userInfo ?? {};

      this.adLaw.setInfo({
        allowed: !!adFormVisible,
        info: adLawInfo,
        status: adLawStatus as AdLawStatus
      });
    }
  };

  featureType = (name: string) => this.featureFlags.get(name);

  setLayoutLoading = (value: boolean) => {
    this.layoutLoading = value;
  };
}
```

```
@import 'shared/styles';
@import './external';
@import './overrides';

:root {
  --typography-link-color: var(--lk-text-link);
  --typography-link-hover-color: var(--lk-text-link-hover);
}

.page-loading {
  display: flex;
  flex: 1;
  align-items: center;
  justify-content: center;
  width: 100%;
  height: 100%;
  min-height: var(--spacing-1900);

  &.page-loading--global {
    position: relative;
    z-index: 15;
    background-color: var(--lk-bg);
  }

  .content & {
    height: 442px;
    min-height: auto !important;
  }

  &.layout_loading {
    position: fixed;
    top: 0;
    left: 0;
    z-index: 99999;
    width: 100%;
    height: 100%;
    background-color: rgb(255 255 255 / 0.15);

    [data-theme='dark'] & {
      background-color: rgb(41 43 55 / 0.15);
    }

    ~ .layout_scroll {
      filter: blur(2px);
    }
  }
}

.app_failed {
  display: flex;
  width: 100%;
  min-height: 100vh;
}

.app_failed-alert {
  margin: auto;
```

```
    a {
      text-decoration: underline;
    }
  }

  .screen-locker {
    overflow: hidden;

    body {
      overflow: hidden;
    }
  }

  import './dashboard.scss';

  import { useEffect } from 'react';
  import { observer } from 'mobx-react-lite';
  import { Loader } from 'tp-design-system';

  import { Permission } from 'entities/user';
  import { Hints } from 'features/hints';
  import { getCookie, gtmEvent, scrollPage, UXS } from 'shared/lib';
  import { $locales, $rootStore } from 'shared/model';
  import { PageTitle } from 'shared/ui';

  import {
    DashboardNews,
    DashboardReferrals,
    DashboardSlider,
    DashboardStatistic,
    EmeraldBlock
  } from './ui';

  export const Dashboard = observer(() => {
    const { language, t } = $locales;
    const { dashboardStore, featureType } = $rootStore;
    const { loading, init, reset } = dashboardStore;

    const showSlider = !getCookie('hide_dashboard_promo');

    useEffect(() => {
      scrollPage(0);

      gtmEvent({
        category: 'Dashboard',
        action: 'open_dashboard'
      });

      return () => reset();
    }, []);

    useEffect(() => {
      document.title = t('pages.dashboard.meta_title');

      init(language);

      if (language === 'en') {
```

```
      UXS.sendEvent('sci_monetize_with_us');
    }
  }, [language]);

  if (loading) {
    return (
      <div className="page-loading">
        <Loader size="2xl" />
      </div>
    );
  }

  return (
    <>
      <PageTitle>{t('pages.dashboard.title')}</PageTitle>

      <div className="dashboard" data-cy="dashboard-page">
        {showSlider && <DashboardSlider />}

        <div className="dashboard-content">
          {featureType('emerald') === 'B' && language === 'en' && (
            <Permission permission="EMERALD_SETTINGS.PARTNER_VIEW">
              <EmeraldBlock />
            </Permission>
          )}

          <Hints />

          <DashboardStatistic />
        </div>

        <div className="dashboard-sidebar">
          <DashboardReferrals />

          <DashboardNews />
        </div>
      </div>
    </>
  );
});


import { makeAutoObservable, runInAction } from 'mobx';

import { HintsStore } from 'features/hints';
import { errorHandler } from 'shared/api';
import type { RootStore } from 'shared/model';
import { $locales, $user } from 'shared/model';
import type { LocaleCode } from 'shared/types';

import { DashboardComputedRows } from './DashboardComputedRows';
import { DashboardComputedTotals } from './DashboardComputedTotals';
import { DashboardSettingsModel } from './DashboardSettingsModel';
import { DashboardTableModel } from './DashboardTableModel';
import { NewsModel } from './NewsModel';
import { DashboardApi } from '../api/dashboard';
import type { DashboardTab, Slide } from '../types/dashboard.types';
```

```
export class DashboardStore {
  private rootStore: RootStore;

  loading: boolean = true;
  slides: Slide[] = [];
  tabs: DashboardTab[] = [];
  news: NewsModel;
  hints: HintsStore;
  tables: Map<string, DashboardTableModel>;
  settings: DashboardSettingsModel;
  shownOptions = ['all', 'pinned', 'with_data'];

  constructor(rootStore: RootStore) {
    makeAutoObservable(this);

    this.rootStore = rootStore;
    this.tables = new Map();
    this.settings = new DashboardSettingsModel(rootStore);
    this.news = new NewsModel();
    this.hints = new HintsStore(rootStore);
  }

  reset = () => {
    this.slides = [];
    this.tables.clear();
    this.loading = true;
  };

  init = async (language: LocaleCode) => {
    try {
      await Promise.all([this.getStatisticsData(language), this.hints.init()]);
    } finally {
      runInAction(() => {
        this.loading = false;
      });
    }
  };

  private getStatisticsData = async (language: LocaleCode) => {
    try {
      const { tabs } = await DashboardApi.getStatistics(language, $user.currency);

      runInAction(() => {
        // init tables models
        tabs.forEach(tab => {
          this.tables.set(tab.name, new DashboardTableModel(tab, this.rootStore));
        });

        // create array of stat tabs
        this.tabs = tabs.map(tab => {
          return {
            name: tab.name,
            value: tab.value ? tab.value.value : 0
          };
        });
      });
```

```
    } catch (error: unknown) {
      errorHandler(error);
    }
  };

  getSliderData = async () => {
    const { email, marker } = $user;

    try {
      const { dashboard_promos } = await DashboardApi.getPromos($locales.language);

      dashboard_promos.map(promo => {
        if (promo.link.includes('typeform.com')) {
          promo.link += `#email=${email}&marker=${marker}`;
        }

        if (promo.link.includes('surveymonkey.com')) {
          promo.link += `?email=${email}&marker=${marker}`;
        }

        return promo;
      });

      runInAction(() => {
        this.slides = dashboard_promos;
      });
    } catch (error: unknown) {
      errorHandler(error);
    }
  };

  get current() {
    const current = this.tables.get(this.settings.visualPeriod);

    if (!current) return null;

    const { rows } = new DashboardComputedRows(current.rows, this.settings);
    const { totals } = new DashboardComputedTotals(rows);

    const rootStore = this.rootStore;

    return {
      totals,
      rows,
      rootStore
    };
  }
}


import { makeAutoObservable, runInAction } from 'mobx';

import { errorHandler } from 'shared/api';
import type { RootStore } from 'shared/model';
import { $locales } from 'shared/model';

import { DashboardApi } from '../api/dashboard';
```

```
import type { DashboardSettings } from '../types/dashboard.types';

export class DashboardSettingsModel {
  private rootStore: RootStore;
  pinned: DashboardSettings['pinned'] = [];
  sort_by: DashboardSettings['sort_by'] = '';
  period: DashboardSettings['period'] = 'month';
  shown: string = 'all';
  sort_order: DashboardSettings['sort_order'] = 'desc';
  visualPeriod: DashboardSettings['period'] = 'month'; // поле для визуализации данных в таблице

  constructor(rootStore: RootStore) {
    this.rootStore = rootStore;
    makeAutoObservable(this);
  }

  set = (settings: DashboardSettings): void => {
    runInAction(() => {
      this.pinned = settings.pinned || [];
      this.sort_by = settings.sort_by || '';
      this.period = settings.period || 'month';
      this.visualPeriod = settings.period || 'month';
      this.shown = settings.shown || 'all';
      this.sort_order = settings.sort_order = 'desc';
    });
  };

  private get settings(): DashboardSettings {
    return {
      pinned: this.pinned,
      sort_by: this.sort_by,
      period: this.period,
      shown: this.shown,
      sort_order: this.sort_order
    };
  }

  private updateSettings = (): void => {
    try {
      DashboardApi.postSettings($locales.language, this.settings);
    } catch (error: unknown) {
      errorHandler(error);
    }
  };

  setShown = (value: string): void => {
    runInAction(() => {
      this.shown = value;
    });

    this.updateSettings();
  };

  handlePin = (id: number): void => {
    if (!this.settings.pinned.includes(id)) {
      runInAction(() => {
        this.pinned = [...this.settings.pinned, id];
```

```
      });
    } else {
      runInAction(() => {
        this.pinned = this.settings.pinned.filter(val => val !== id);
      });
    }

    this.updateSettings();
  };

  setSortBy = (key: string): void => {
    let sortChange: 'desc' | 'asc';

    if (this.settings.sort_by === key) {
      sortChange = this.settings.sort_order === 'desc' ? 'asc' : 'desc';
    }

    runInAction(() => {
      this.sort_by = key;
      this.sort_order = sortChange || 'desc';
    });

    this.updateSettings();
  };

  changePeriod = (period: string): void => {
    runInAction(() => {
      this.period = period;
    });

    setTimeout(() => {
      runInAction(() => {
        this.visualPeriod = period;
      });
    }, 300);

    this.updateSettings();
  };
}
```