САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: “Фронт-энд разработка”

Отчет
Лабораторная работа №2

Выполнил: Митурский Богдан Антонович

Группа: K33392

Санкт-Петербург

2023 г.

Цель:

Реализовать взаимодействие главной страницы с API сервера игры для получения состояния сундуков на главном экране опираясь на данные пользователя.

Программное обеспечение: TypeScript, Node Typescript, Socket.io, React

Ход работы:

1. Реализуем функцию request для проекта:

```
import axios from "axios";
import axiosRetry from "axios-retry";
import { SERVER_URL } from "@/config/constants";
import getAuthParams from "./getAuthParams";

axiosRetry(axios, {
  retries: 180,
  retryDelay: (retryCount) => {
    let retryTimeout = retryCount * 500;
    if (retryTimeout > 1500) {
      retryTimeout = 1500;
    }
    return (retryTimeout = 1500);
  },
});

export async function request<T>({
  auth = true,
  method,
  params,
  controller,
  ignoreErrors,
}: {
  method: string;
  auth?: boolean; // Нужна ли авторизация
  params?: { [key: string]: any };
  controller?: AbortController;
  ignoreErrors?: boolean;
}) {
  const headers: { Authorization?: string } = {};
  if (auth) {
    headers.Authorization = `${getAuthParams()}`;
  }
  const response = await axios.get<T>(`${SERVER_URL}/${method}`, {
    headers,
    params,
```

```
    signal: controller?.signal,
    validateStatus: ignoreErrors ? () => true : undefined,
  });
  return response.data;
}
```

2. Реализуем функцию для запроса базовых данных пользователя:

```
export default async function getUser(callback: (user: UserState) => void) {
  const userId = Number(getId());
  if (!userId) return;

  const user = await request({ auth: true, method: "user/get" });
  if (!user) return;

  callback({
    ...user
  } as UserState);
}
```

3. Сохраним данные пользователя в redux сторе для доступа из любой части приложения:

```
import { createAsyncThunk, createSlice, PayloadAction } from
"@reduxjs/toolkit";
import { IChestTypes } from "@/panels/ChestOpening/prize";
import { CardRarities } from "@/data/cards";
import { request } from "@/utils/request";

type StatsState = {
  wins: number;
  losses: number;
  draws: number;
};

type Bonuses = {
  daily: number;
  subGroup: boolean;
  joinChat: boolean;
  notificationApp: boolean;
  notificationBot: boolean;
  addToFavorites: boolean;
  rewardedGift: {
    lastUpdate: number;
    adWatchedTimes: number;
  };
```

```ts
};

type HistoryState = {
  id: number;
  userResponse: {
    rating: number;
    chest: IChestTypes;
    status: "win" | "lose" | "draw";
  };
  opponentInfo: {
    id: number;
    photo_200: string;
  };
  date: number;
};

export type IChest = {
  type: IChestTypes;
  willOpenAt?: number;
  status: "opening" | "idle";
  adWatchedTimes: number;
} | null;

export type EntityType =
  | "bigSheep"
  | "crashedPlane"
  | "deadFarm"
  | "doubleFarm"
  | "farm"
  | "fastSheep"
  | "pasture"
  | "sheep"
  | "empty1"
  | "empty2"
  | "empty3"
  | "empty4";

export type DeckItem = { entity: EntityType } & (
  | {
      blocked: true;
      part: number;
      partMax?: number; // Для элемента карты
    }
  | {
      blocked?: false;
    }
);
```

```ts
export type Lottery = {
  endsAt: number;
  postLink: string;
  imageLink: string;
} | null;

export type UserState = {
  _id: number;
  name: string;
  balance: number;
  stats: StatsState;
  deck: [DeckItem, DeckItem, DeckItem, DeckItem];
  photo_200: string;
  history: HistoryState[] | [];
  collection: DeckItem[];
  isToken?: boolean;
  chests: [IChest, IChest, IChest, IChest];
  lobby?: {
    _id: number;
    url: string;
  };
  bonuses: Bonuses;
  rating: number;
  isTutorialPassed: boolean;
  shop: IShop;
  isShopLoading: boolean;
  lottery?: Lottery;
};

export type Coin = Extract<
  StoreProducts,
  StoreProducts.coins | StoreProducts.heapCoins | StoreProducts.bigHeapCoins
>;

export interface IShop {
  boughtProducts: StoreProducts[];
  chest: { chestType: IChestTypes | null; currency: "voices" | "coins" };
  cardFragment: { card: string | null; currency: "voices" | "coins" };
  prices: {
    coins: Record<Coin, number>;
    chests: Record<IChestTypes, { coins: number; voices: number }>;
    cards: Record<CardRarities, { coins: number; voices: number }>;
  };
  nextShopUpdate: number;
  promotion?: {
    title: string;
    description: string;
    endsAt: number;
```

```
    discounts: {
      voices: number;
      coins: number;
    };
  };
  serverDateNow: number;
}

export type ShopPrices = IShop["prices"];

const initialState: UserState = {
  _id: 0,
  name: "",
  photo_200: "",
  balance: 0,
  history: [],
  stats: {
    wins: 0,
    losses: 0,
    draws: 0,
  },
  deck: [
    { entity: "empty1" },
    { entity: "empty2" },
    { entity: "empty3" },
    { entity: "empty4" },
  ],
  collection: [],
  chests: [null, null, null, null],
  bonuses: {
    daily: 0,
    subGroup: false,
    joinChat: false,
    notificationApp: false,
    notificationBot: false,
    addToFavorites: false,
    rewardedGift: {
      lastUpdate: 0,
      adWatchedTimes: 0,
    },
  },
  rating: 0,
  isTutorialPassed: false,
  shop: {
    boughtProducts: [],
    chest: { chestType: null, currency: "coins" },
    cardFragment: { card: null, currency: "coins" },
    prices: {
```

```
      coins: {
        coins: 0,
        heapCoins: 0,
        bigHeapCoins: 0,
      },
      chests: {
        common: { coins: 0, voices: 0 },
        magic: { coins: 0, voices: 0 },
        legendary: { coins: 0, voices: 0 },
      },
      cards: {
        common: { coins: 0, voices: 0 },
        rare: { coins: 0, voices: 0 },
        epic: { coins: 0, voices: 0 },
        legendary: { coins: 0, voices: 0 },
      },
    },
    nextShopUpdate: 0,
    serverDateNow: Date.now(),
  },
  isShopLoading: false,
  lottery: undefined,
};

export enum StoreProducts {
  discontCard = "discontCard",
  discontChest = "discontChest",
  magicChest = "magicChest",
  commonChest = "commonChest",
  legendaryChest = "legendaryChest",
  coins = "coins",
  heapCoins = "heapCoins",
  bigHeapCoins = "bigHeapCoins",
}

// Функция для обработки статуса сундука в связи с нуждами интерфейса
const transfromOpenTime = (obj: IChest[]) => {
  const now = Date.now();
  obj.forEach((chest) => {
    if (!chest || chest.status !== "opening" || chest.willOpenAt ===
undefined)
      return;
    chest.willOpenAt += now;
  });
};

const UserSlice = createSlice({
  name: "User",
```

```
  initialState,
  reducers: {
    setUser(state: any, action: PayloadAction<any>) {
      if (action.payload?.chests) transfromOpenTime(action.payload.chests);
      Object.keys(action.payload).forEach((key) => {
        state[key as keyof UserState] = action.payload[key as keyof
UserState];
      });
    },
    setBalance(state: UserState, action: PayloadAction<number>) {
      state.balance = action.payload;
    },
    setBonuses(state: UserState, action: PayloadAction<Bonuses>) {
      state.bonuses = action.payload;
    },
    setDeck(
      state: UserState,
      action: PayloadAction<[DeckItem, DeckItem, DeckItem, DeckItem]>
    ) {
      state.deck = action.payload;
    },
    setCollection(state: UserState, action: PayloadAction<DeckItem[]>) {
      state.collection = action.payload;
    },
    setChestStatus(
      state: UserState,
      action: PayloadAction<{
        index: number;
        status: "empty" | "opening";
      }>
    ) {
      const chest = state.chests[action.payload.index];
      if (chest === null) return;

      if (action.payload.status === "opening") {
        state.chests[action.payload.index] = {
          status: action.payload.status,
          type: chest.type,
          adWatchedTimes: chest.adWatchedTimes,
        };
      }

      if (action.payload.status === "empty") {
        state.chests[action.payload.index] = null;
      }
    },
    setChests(state: UserState, action: PayloadAction<UserState["chests"]>) {
      transfromOpenTime(action.payload);
```

```
    state.chests = action.payload;
  },
  changeDeck(
    state: UserState,
    action: PayloadAction<{ index: number; card: DeckItem }>
  ) {
    state.deck[action.payload.index] = action.payload.card;
  },
  changeCollection(
    state: UserState,
    action: PayloadAction<{ index: number; card: DeckItem }>
  ) {
    state.collection[action.payload.index] = action.payload.card;
  },
  setUserLobby(
    state: UserState,
    action: PayloadAction<{ url: string; _id: any }>
  ) {
    state.lobby = action.payload;
  },
  completeTutorial(state: UserState) {
    state.isTutorialPassed = true;
  },
  startTutorial(state: UserState) {
    state.isTutorialPassed = false;
  },
  setShop(state: UserState, action: PayloadAction<IShop>) {
    state.shop = action.payload;
    if (action.payload.nextShopUpdate)
      state.shop.nextShopUpdate += Date.now();
  },
  setDiscontProducts(
    state: UserState,
    action: PayloadAction<{
      chest: IShop["chest"];
      cardFragment: IShop["cardFragment"];
    }>
  ) {
    state.shop.chest = action.payload.chest;
    state.shop.cardFragment = action.payload.cardFragment;
  },
  setBoughtProducts(
    state: UserState,
    action: PayloadAction<StoreProducts[]>
  ) {
    state.shop.boughtProducts = action.payload;
  },
  setShopLoading(state: UserState, action: PayloadAction<boolean>) {
```

```
      state.isShopLoading = action.payload;
    },
    setLottery(state: UserState, action: PayloadAction<Lottery>) {
      state.lottery = action.payload;
      if (state.lottery) {
        state.lottery.endsAt += Date.now();
      }
    },
  },
  extraReducers: (builder) => {
    builder
      .addCase(fetchShop.pending, (state) => {
        state.isShopLoading = true;
      })
      .addCase(fetchShop.fulfilled, (state) => {
        state.isShopLoading = false;
      });
  },
});

export default UserSlice.reducer;
export const {
  setUser,
  setBalance,
  setDeck,
  setCollection,
  changeDeck,
  changeCollection,
  setChestStatus,
  setChests,
  setUserLobby,
  setBonuses,
  completeTutorial,
  setShop,
  setShopLoading,
  setBoughtProducts,
  setDiscontProducts,
  setLottery,
  startTutorial,
} = UserSlice.actions;
```

4. Добавим в компонент сундуков код получения данных из стора и пробрасывания в сундуки:

```
const Chests = () => {
  const dispatch: RootDispatch = useDispatch();
  const { chests } = useSelector((state: RootState) => state.User);
  const imageRefs = useRef<HTMLDivElement[]>([]);
```

```
  const [loadingIndex, setLoadingIndex] = useState<number | null>(null);

  return (
    <Root>
      {chests.map((chest, i) => (
        <Chest
          data-tutorial-step={!i ? DataTutorialIds.Chest : -1}
          loading={loadingIndex === i}
          available={loadingIndex === null}
          data={chest}
          key={i}
          onClick={() => handleChestClick(chest, i)}
          imageRef={(ref) => (imageRefs.current[i] = ref)}
        />
      ))}
    </Root>
  );
};

export default React.memo(Chests);
```

5. В сами сундуки добавим работу с передаваемой информацией. Будем опираться на willOpenAt чтобы понять статус сундука (закрыт / открывается / открыт):

```
const Chest = ({
  data,
  imageRef,
  loading,
  available,
  ...props
}: {
  loading: boolean;
  data: IChest;
  available: boolean;
  imageRef?: (arg1: HTMLImageElement) => HTMLImageElement | void;
} & React.HTMLAttributes<HTMLDivElement>) => {
  const getTimeString: (config?: { ad?: boolean }) => {
    status?: IStatus;
    time: string;
  } = (config) => {
    if (!data) return { status: undefined, time: ". . ." };
    if (data.status !== "idle" && data.willOpenAt === undefined)
      return { status: "opening", time: ". . ." };

    const openingTime =
```

```
    (data.status === "idle"
      ? Date.now() + ChestsData[data.type].openingTime + 5000
      : data.willOpenAt || 0) - Date.now();
  const d = config?.ad
    ? Math.floor(ChestsData[data.type].openingTime / (6 * 60 * 1000)) *
      60 *
      1000
    : Math.max(0, openingTime);
  if (d <= 0) return { status: "ready", time: ". . ." };

  const hours = Math.floor(d / 60 / 60 / 1000);
  const minutes = Math.floor((d - hours * 60 * 60 * 1000) / 60 / 1000);
  let time = `${hours} ч. ${minutes} м.`;

  if (hours + minutes === 0) {
    time = "< 1 мин.";
  } else if (hours === 0) {
    time = `${minutes} мин.`;
  } else if (minutes === 0) {
    time = `${hours} ${formatWordByNumber(hours, "час", "часа", "часов")}`;
  }

  return {
    status: data.status,
    time,
  };
};

const initialTime = useMemo(() => getTimeString(), []);
const [status, setStatus] = useState(initialTime.status);
const [time, setTime] = useState(initialTime.time);
const adWatchBonus = useMemo(
  () => getTimeString({ ad: true }).time,
  [data?.type]
);

const timeTitleRef = useRef();
const lastData = useRef<IChest | null>(null);

useEffect(() => {
  const interval = {} as { current?: NodeJS.Timer };

  const update = () => {
    const { status, time } = getTimeString();

    setStatus(status);
    setTime(time);
    if (status !== "opening") clearInterval(interval.current);
```

```
      return { status, time };
    };

    const { status } = update();
    if (status !== "opening") return;

    if (
      lastData.current &&
      data &&
      timeTitleRef.current &&
      lastData.current.willOpenAt !== data.willOpenAt
    ) {
      setTimeout(() => {
        anime({
          targets: timeTitleRef.current,
          keyframes: [
            { scale: 1.23, rotate: -1 },
            { scale: 0.93, rotate: 2 },
            { scale: 1, rotate: 0 },
          ],
          duration: 500,
          autoplay: true,
          easing: "cubicBezier(0.690, 0.545, 0.105, 1.460)",
        });
      }, 0);
    }
    lastData.current = data;

    interval.current = setInterval(update, 1000);
    return () => clearInterval(interval.current);
  }, [data]);

  const skipButton = !!data && status === "opening" && data.adWatchedTimes <
3;

  const chestImage = data ? ChestsImages[data.type]?.image : undefined;

  const clickable =
    available &&
    (status === "opening"
      ? skipButton
      : status === "idle" || status === "ready");

  return (
    <Wrapper
      status={status}
      skipButton={skipButton}
```

```
      loading={loading}
      hasActive={clickable}
      hasHover={clickable}
      clickable={clickable}
      activeMode="clicked"
      className={loading ? "clicked" : ""}
      {...props}
    >
      <Content status={status}>
        {data === null && <div>{"Место\nдля\nсундука"}</div>}

        {data && status === "idle" && (
          <>
            <Image ref={imageRef} src={chestImage} status={status} />
            <BorderedText>{time}</BorderedText>
          </>
        )}

        {data && status === "opening" && (
          <>
            <BorderedTextSmall ref={timeTitleRef}>{time}</BorderedTextSmall>
            <Image ref={imageRef} src={chestImage} status={status} />
            {skipButton && (
              <SkipButton
                available={available}
              >{`Ускорить -${adWatchBonus}`}</SkipButton>
            )}
          </>
        )}

        {data && status === "ready" && !loading && (
          <>
            <ImageGlowWrapper>
              <Image ref={imageRef} src={chestImage} status={status} />
              <Glow src={glow} />
            </ImageGlowWrapper>
            <BorderedText>Открыть</BorderedText>
          </>
        )}

        {loading && <Spinner />}
      </Content>
    </Wrapper>
  );
};

export default Chest;
```

Вывод:

В ходе работы я научился работать с внешним api, подключил к нему фронтенд, запросил данные пользователя и пробросил информацию о сундуках пользователя в интерфейс. Итоговая панель с сундуками выглядит следующим образом:

Данные полученные из api:


```
▼ 0: {empty: false, adWatchedTi
   adWatchedTimes: 0
   empty: false
   status: "idle"
   type: "common"
▼ 1: {type: "magic", status: "o
   adWatchedTimes: 0
   status: "opening"
   type: "magic"
   willOpenAt: -25252318
▼ 2: {type: "legendary", status
   adWatchedTimes: 0
   status: "opening"
   type: "legendary"
   willOpenAt: -4999669078
▼ 3: {type: "common", status: "
   adWatchedTimes: 0
   status: "opening"
   type: "common"
   willOpenAt: 10742488
```