

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по производственной практике
Тема: Разработка сервиса авторизации приложения “WarThunder
Assistant”

Студент гр. 8382

Мирончик П.Д.

Преподаватель

Петросов С.Ю.

Санкт-Петербург

2021

ЗАДАНИЕ НА ПРОИЗВОДСТВЕННУЮ ПРАКТИКУ

Студент Мирончик П.Д.

Группа 8382

Тема практики: Разработка сервиса авторизации приложения “War Thunder Assistant”

Задание на практику: разработать сервис авторизации, позволяющий авторизовать пользователя из приложения WarThunder Assistant с использованием SSO сервера. Необходимо поддерживать возможность двухфакторной аутентификации с использованием логина, пароля и кода подтверждения, а также Steam авторизации через протокол OpenID.

Сроки прохождения практики: 30.06.2021 - 20.07.2021

Дата сдачи отчета: 20.07.2021

Дата защиты отчета: 20.07.2021

Студент гр. 8382

Мирончик П.Д.

Преподаватель

Петросов С.Ю.

АННОТАЦИЯ

Работа представляет собой внедрение сервиса авторизации в уже существующее приложение на языке Dart, с использованием фреймворка Flutter. Сервис авторизации имеет возможность авторизовать пользователя при помощи логина, пароль и двухфакторного кода аутентификации (если та настроена у пользователя), а также Steam OpenID авторизации в качестве аналогичного способа входа. Сервис должен корректно реагировать на любые события: отключение интернета, неправильный ответ сервера, ошибки сервера, связанные с проблемами авторизации конкретного пользователя (неверные данные для входа, заморозка аккаунта и т.п.), и отображать текущее состояние авторизации пользователю при помощи интерфейса.

SUMMARY

The work is the implementation of an authorization service into an existing application in the Dart language, using the Flutter framework. The authorization service has the ability to authorize a user using a username, password and two-factor authentication code (if configured for the user), as well as Steam OpenID authorization as a similar login method. The service must correctly respond to any events: Internet disconnection, incorrect server response, server errors associated with authorization problems for a particular user (incorrect login information, account freeze, etc.), and display the current authorization status to the user using the interface.

СОДЕРЖАНИЕ

	Введение	5
1.	Подготовка к работе	6
1.1.	Основные понятия и структуры	6
1.2.	Алгоритм авторизации	7
1.2.1.	Авторизация через пару логин-пароль	7
1.2.2.	Авторизация через Steam OpenID	8
1.2.3.	Получение профиля	9
1.2.4.	Повторная авторизация	9
2.	Реализация	10
2.1.	Базовые классы	10
2.1.1.	LoginState	10
2.1.2.	LoginSession	11
2.1.3.	LoginService	12
2.1.4.	LoginListener	14
2.1.5.	LoginScreen	14
2.1.6.	LoginError	15
2.2.	Состояния	16
2.2.1.	Дерево состояний	16
2.2.2.	StateEnterLogin	17
2.2.3.	StateAgreements	18
2.2.4.	StateAgreementDetails	19
2.2.5.	StateEnterPassword	19
2.2.6.	StateEnterCode	21
2.2.7.	StateSteamFetchUrl	23
2.2.8.	SteamWebViewState	24
2.2.9.	StateSteamLogin	25
	Заключение	27

ВВЕДЕНИЕ

Разработка сервиса велась на языке `Dart` с использованием фреймворка `Flutter` для взаимодействия с системой устройства, в том числе визуализации.

Сервис авторизации должен быть достаточно простым и надежным, чтобы уменьшить количество возможных ошибок при его разработке, и в то же время сделать процесс взаимодействия с ним понятным и безопасным. Поэтому в качестве архитектуры сервиса была выбрана машина состояний.

Итоговая версия сервиса получилась достаточно объемной, поэтому описание реализации отдельных классов, за исключением базовых, определяющих архитектуру, опущено, с целью сократить объем отчета. В разделе «Подготовка к работе» описана теория, более развернуто описывающая поставленную задачу. Здесь же описаны определения и сокращения, которые используются при дальнейшем описании работы. Раздел «Базовые классы» описывает основу архитектуры сервиса. В разделе «Состояния» находится описание состояний и экранов с подробным объяснением всех возможных переходов между ними.

1. ПОДГОТОВКА К РАБОТЕ

1.1. Основные понятия и структуры

Целью авторизации является получение идентификационных данных и получение игрового профиля пользователя. Важно понимать, что их получение производится двумя разными запросами на сервер, и получение идентификационных данных не гарантирует получения профиля, а авторизация считается успешной только в случае получения и профиля, и данных.

Идентификационные данные - информация об аккаунте пользователя, используемая в служебных целях для идентификации пользователя, формирования, подписи запросов:

```
class User {  
    String name;  
    String token;  
    String userId;  
    String deviceId;  
    String login;  
    String password;  
}
```

name - ник пользователя;

token - jwt токен аутентификации;

userId - уникальный постоянный идентификатор пользователя;

deviceId - идентификатор устройства, с которого проведена авторизация;

login - логин пользователя;

password - пароль пользователя;

Данные игрового профиля (Profile) - набор информации о пользователе в игре WarThunder: статистика, игровая валюта, список друзей, список техники и т.п. В модели профиля содержится несколько десятков полей, разбирать которые в данной работе не имеет смысла, т.к. они не используются при выполнении задачи.

Protobuf - формат сериализации данных, используемый в данном случае для передачи данных профиля пользователя.

Two Factor Authorization (TFA) - двухфакторная авторизация.

WTA - сокращение от War Thunder Assistant.

GaijinPass - приложение для авторизации, позволяющее подтвердить вход при двухфакторной авторизации. Подробнее взаимодействие WTA и GaijinPass см. в разделе 1.2.1.

1.2. Алгоритм авторизации

Предусматривается два вида авторизации: используя логин и пароль и через Steam OpenID. Отметим, что авторизация представляет собой получение идентификационных данных (разделы 1.2.1 и 1.2.2), а затем получение профиля пользователя на основании этих данных (раздел 1.2.3). Если какой-либо из этапов проваливается, авторизация также считается провалившейся.

1.2.1. Авторизация через пару логин-пароль

Пользователь вводит логин аккаунта. Затем пользователь должен принять лицензионное соглашение, если пользователь с таким логином не принял соглашение ранее на этом устройстве. На этом этапе каких-либо проверок, кроме валидности логина (известно, что логин должен иметь формат email), не предусмотрено. После принятия лицензионного соглашения пользователь вводит пароль и производится попытка входа. Если аккаунт с такой парой логин-пароль найден, то возможна необходимость подтвердить вход, используя код двухфакторной авторизации, для аккаунтов с активированной двухфакторной авторизацией.

Подтвердить вход пользователь может двумя способами.

1. Ввод кода TFA (двухфакторной авторизации) в предложенное поле.
2. Использовать приложение GaijinPass для подтверждения входа. В этом случае пользователь должен перейти в приложение GaijinPass и нажать кнопку «Подтвердить» в появившемся окне. Для использования такого

способа авторизации необходимо отправить запрос на сервер авторизации, содержащий идентификатор пользователя и сессии авторизации с таймаутом в одну минуту. В течении минуты может быть получен либо положительный ответ с кодом TFA, либо сообщение об отклонении запроса на авторизацию; отсутствие ответа в установленный срок означает, что пользователь не произвел никакого действия и можно послать еще один запрос на получение кода.

Отдельно отметим возможность использования кода из самого приложения War Thunder Assistant и приложения GaijinPass, установленного на этом же устройстве. Если к WTA или к установленному на этом же устройстве GaijinPass привязан пользователь, то код может быть получен автоматически с использованием SDK Android.

1.2.2. Авторизация через Steam OpenID

Steam OpenID авторизация состоит из трех этапов.

1. Получение ссылки на страницу авторизации. Для получения этой ссылки выполняется отдельный запрос к серверу приложения.

2. Авторизация на странице Steam OpenID. Во внутреннем браузере по полученной в п.1 ссылке открывается страница авторизации, где пользователь должен войти, используя данные своего Steam аккаунта. После этого перехватывается событие переадресации протокола OpenID и данные из запроса используются для дальнейшей авторизации.

3. Выполняется передача полученных в п.2 данных на сервер приложения, на основании которых производится идентификация пользователя и сервис авторизации получает данные аккаунта пользователя.

1.2.3. Получение профиля

Обязательный этап авторизации - получение профиля пользователя. И Steam авторизация, и двухфакторная авторизация через логин-пароль позволяют получить идентификационные данные аккаунта. После их получения всегда следует запрос на получение профиля. Если получить

профиль не удастся по техническим причинам (проблемы с сетью, сервером и т.п.), авторизация прерывается и возвращается на этап ввода пароля. Возможны также различные предусмотренные ошибки: бан аккаунта и его заморозка. В случае получения этих ошибок авторизация также прерывается и возвращается на этап ввода логина. Если получение профиля было успешным, процесс авторизации заканчивается.

1.2.4. Повторная авторизация

Если обнаруживается, что выданный при прошлой авторизации токен закончился, и авторизация проводилась с использованием пары логин-пароль, то сервис авторизации проводит повторную авторизацию, которая выполняется незаметно для пользователя, без открытия какого-либо диалогового окна. Если авторизация не удалась (по причине плохого соединения, изменившихся данных или при необходимости дополнительно передать код TFA), открывается окно сервиса авторизации на этапе ввода пароля или кода TFA, в зависимости от причины неудачи.

2. РЕАЛИЗАЦИЯ

2.1. Базовые классы

Рассмотрим список классов, составляющих основу сервиса.

2.1.1. LoginState

```
abstract class LoginState {  
    LoginSession session;  
  
    @mustCallSuper  
    void onStart(LoginSession session) {}  
  
    @mustCallSuper  
    void onStop() {}  
  
    void onWillPop();  
}
```

Базовый класс состояния. Методы `onStart` и `onStop` позволяют отслеживать, является ли объект состояния текущим и проводить инициализацию и освобождение ресурсов. Каждое состояние должно определить метод `onWillPop`, которое является событием отмены и вызывается при нажатии системной кнопки «Назад».

Состояние плотно взаимодействует со связанным объектом сессии `session`. Через нее осуществляется установка виджета интерфейса, виджета ошибки, установка нового состояния или завершение сессии с определенным результатом.

<code>void onStart(LoginSession session)</code>	Метод инициализации состояния. Гарантируется, что, начиная с вызова <code>onStart</code> и до вызова <code>onStop</code> , объект сессии <code>session</code> будет доступен.
<code>void onStop()</code>	Метод остановки состояния. При вызова этого метода состояние должно завершить все операции и освободить ресурсы.
<code>void onWillPop()</code>	Метод, вызываемый при возникновении системного события «Назад». Каждое

состояние должно корректно обрабатывать этот вызов, не допуская заикливания на одном состоянии.

2.1.2. LoginSession

```
class LoginSession {
    LoginState _currentState;
    LoginError _error;
    LoginScreenState _screen;

    LoginSession(this._currentState, {LoginError error}) : _error
    = error;

    factory LoginSession.SSO({LoginError error}) =>
    LoginSession(StateEnterLogin(), error: error);

    void start() {...}

    void finish(bool success, [User user]) {...}

    void cancel() {...}

    void setScreen(Widget screen) {...}

    void setError(LoginError error) {...}

    void setState(LoginState state) {...}

    void onScreenInit(LoginScreenState screenState) {...}

    void onWillPop() {...}

    void onScreenDispose() {...}
}
```

`LoginSession` является основным объектом сессии, хранит текущее состояние, осуществляет переход между состояниями (вызывая `onStop` у старого состояния и `onStart` у нового), а также реализует связь между состоянием и виджетом интерфейса.

<code>void start()</code>	Запускает сессию. Создает виджет <code>LoginScreen</code> и устанавливает первое состояние <code>StateEnterLogin</code> без вызова <code>onStart</code> (см. <code>onScreenInit</code>).
<code>void finish(bool</code>	Завершает сессию с результатом

<code>success, [User user])</code>	success: ошибка либо успешная авторизация. В случае успешной авторизации параметр user должен быть не пустым.
<code>void cancel()</code>	Отмена авторизации без результата и уведомления.
<code>void setScreen(Widget screen)</code>	Устанавливает переданный widget в созданный на старте LoginScreen.
<code>void setError(LoginError error)</code>	Устанавливает виджет ошибки в LoginScreen.
<code>void setState(LoginState state)</code>	Устанавливает новое состояние. Старое состояние гарантированно будет остановлено вызовом LoginState.onStop, новое будет проинициализировано через LoginState.onStart.
<code>void onScreenInit(LoginScreenState screenState)</code>	Вызывается при инициализации созданного виджета LoginScreen. При инициализации также выполняется старт текущего состояния LoginState.onStart.
<code>void onWillPop()</code>	Вызывается привязанным виджетом LoginScreen при системном вызове «Back». Вызывает onWillPop у текущего состояния.
<code>void onScreenDispose()</code>	Вызывается при удалении связанного LoginState из дерева виджетов. Используется для валидации: на этом этапе процесс авторизации должен быть завершен; если это не так - сессия отменяется.

2.1.3. LoginService

```
class LoginService {
  static LoginService instance = ...;

  void login({Widget error}) {...}

  void onReLogin(User user) {...}

  void notifyLoggedIn(User user) {...}

  void logout() {...}

  void subscribe(LoginListener listener) {...}

  void unsubscribe(LoginListener listener) {...}
```

```
}
```

LoginService построен по принципу Single Instance и является классом, который осуществляет связь сервиса с другими частями приложения. Предоставляет возможность подписаться на изменения при помощи слушателя LoginListener через методы subscribe и unsubscribe. Управление авторизацией выполняется вызовом методов login и logout.

Релогин (повторная авторизация при истечении срока действия токена) вынесен в нативную часть приложения. Это связано с тем, что теоретически возможна ситуация, когда Flutter часть приложения неактивна, а авторизованные запросы используются в том числе некоторыми нативными экранами. Поэтому фактически сессия авторизации запускается следующим образом:

```
LoginService.login > "Native relogin" > LoginService.onReLogin > "session start"
```

В случае, если повторная авторизация прошла успешно, запуска сессии не происходит и пользователь не видит экран авторизации и не получает каких-либо лишних сообщений.

<code>static LoginService instance</code>	Экземпляр класса LoginService.
<code>void login({Widget error})</code>	Запуск авторизации. Непосредственно при вызове метода открытия экрана авторизации не происходит, предварительно выполняется попытка релогина.
<code>void onReLogin(User user)</code>	Метод, вызываемый из натива при завершении процесса релогина. Параметр user показывает результат операции и, в случае успеха, устанавливается как результат авторизации.
<code>void notifyLoggedIn(User user)</code>	Вызывается для завершения процесса авторизации с удачным или неудачным результатом, в зависимости от параметра user. При вызова метода текущая сессия (если она была создана) завершается и слушателям сервиса отправляется уведомление о результате.

<code>void logout()</code>	Выполняет выход текущего пользователя. Если в момент вызова активна сессия авторизации, она завершается. При вызове метода <code>logout</code> уведомление о выходе пользователя отправляется в любом случае, даже если пользователь не был авторизован.
<code>void subscribe(LoginListener listener)</code>	Подписывает слушателя <code>listener</code> на уведомления о входах и выходах пользователя.
<code>void unsubscribe(LoginListener listener)</code>	Отписывает слушателя <code>listener</code> на уведомления о входах и выходах пользователя.

2.1.4. LoginListener

```
class LoginListener {
    final Function onLogin;
    final Function onLogout;

    LoginListener({
        this.onLogin,
        this.onLogout
    });
}
```

Слушатель, который можно подписать на события входа и выхода сервиса авторизации `LoginService` через методы `subscribe` и `unsubscribe`. Передаваемые функции `onLogin` и `onLogout` вызываются при входе и выходе пользователя соответственно.

2.1.5. LoginScreen

```
class LoginScreen extends StatefulWidget {
    final LoginSession session;

    LoginScreen(this.session);

    @override
    State createState() => LoginScreenState();
}

class LoginScreenState extends State<LoginScreen> {
    set loginWidget(Widget loginWidget) {...}

    set errorWidget(LoginError errorWidget) {...}
}
```

```
}
```

Виджет, в котором отображается интерфейс сервиса авторизации. При помощи сеттеров `loginWidget` и `errorWidget` связанный объект `LoginSession` устанавливает основной виджет и виджет ошибки соответственно. Виджет ошибки показывается над основным виджетом и может перехватить событие «Назад».

2.1.6. LoginError

```
abstract class LoginError extends Widget{  
    onWillPop(LoginSession session);  
}
```

Базовый класс для виджета ошибок, устанавливаемых в процессе авторизации (см `LoginSession.setError`). Используется как интерфейс, позволяющий определить обработчик события «Назад».

2.2. Состояния

2.2.1. Дерево состояний

Ниже приведено дерево состояний авторизации.

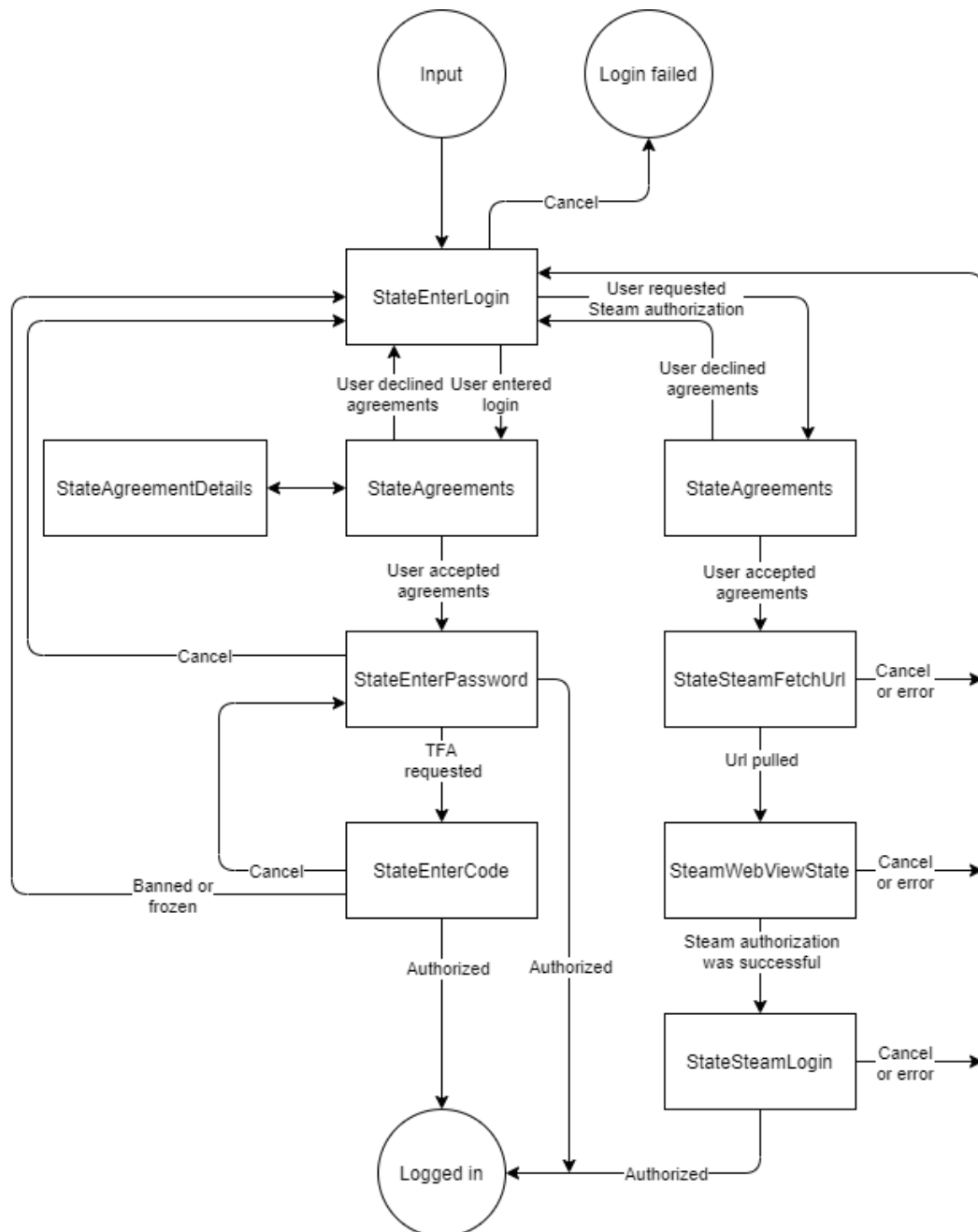


Рис.1. Дерево состояний

2.2.2. StateEnterLogin

Экран, содержащий поле для ввода логина и возможность выбора пользователем Steam авторизации. Из состояния возможны три перехода:

1. Выход из авторизации по сигналу «Назад».

2. Переход в состояние StateAgreements по нажатию кнопки «Далее», с дальнейшим переходом в StateEnterPassword. Если текст в поле ввода логина не соответствует формату « $^[\wedge][\wedge@]+\wedge[\wedge@]+[\wedge@]+\$$ », кнопка «Далее» будет неактивна.

3. Переход в состояние StateAgreements с дальнейшим переходом в StateSteamFetchUrl.

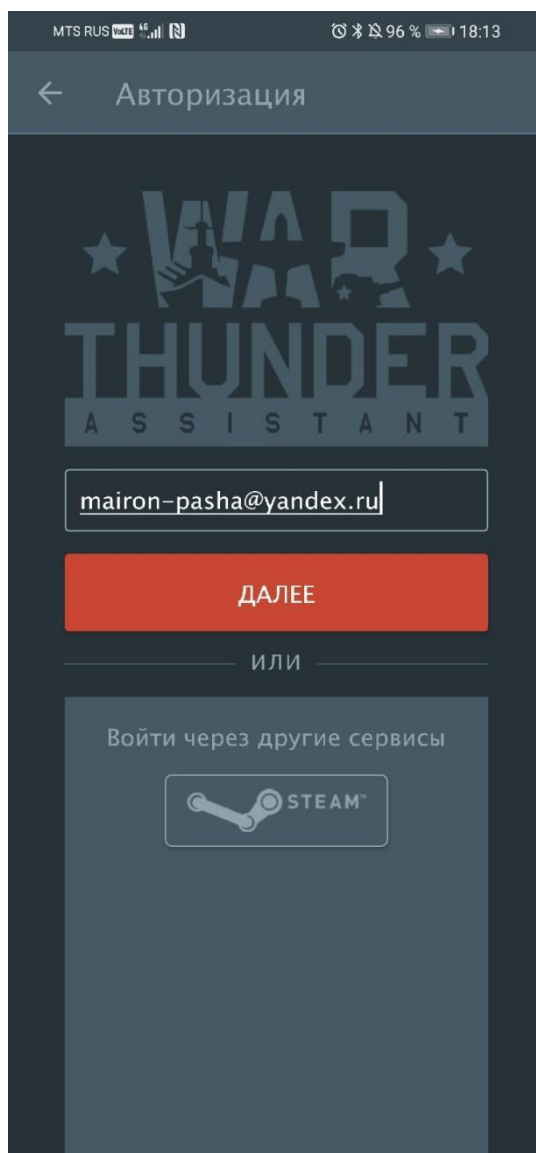


Рис.2. Экран ввода логина

2.2.3. StateAgreements

Экран лицензионного соглашения. Возможные действия:

1. Пользователь может принять соглашение и перейти на следующий экран (`StateEnterPassword` или `StateSteamFetchUrl`) по кнопке «Принять».
2. Пользователь может отклонить соглашение и вернуться на экран `StateEnterPassword`.
3. Пользователь может открыть текст лицензионного соглашения или политики конфиденциальности(экран `StateAgreementDetails`), нажав на одну из ссылок.

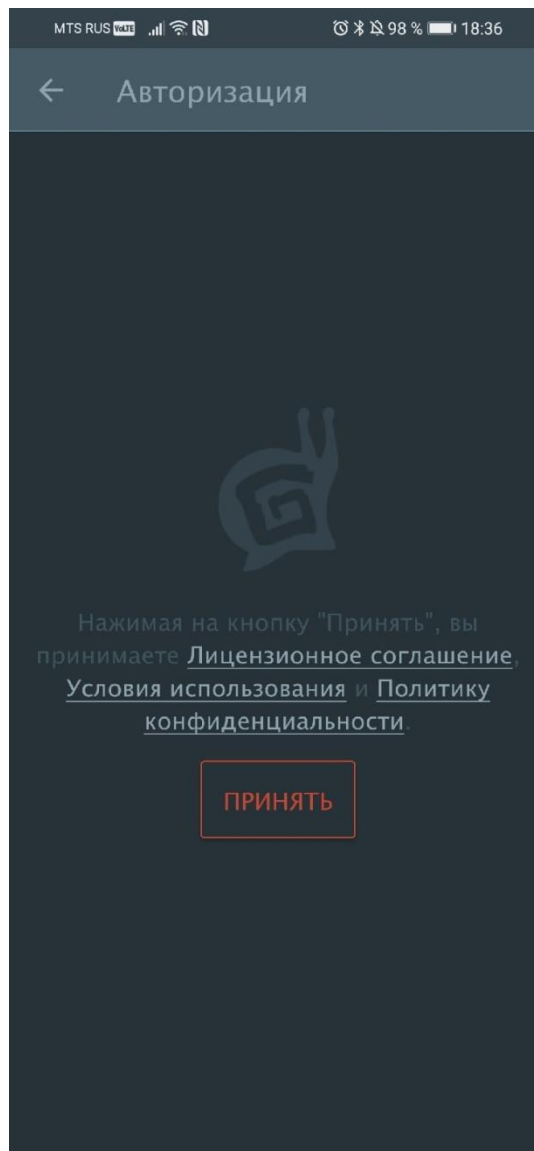


Рис.3. Экран лицензионного соглашения

2.2.4. StateAgreementDetails

Текст юридического документа. Отправляет GET запрос на сервер для получения текста, после чего выводит его на экран. В случае ошибки загрузки показывает соответствующее сообщение с кнопкой «Повторить попытку», нажатие на которую приводит к повторной попытке запросить текст. По событию «Назад» открывается предыдущий экран StateAgreements.

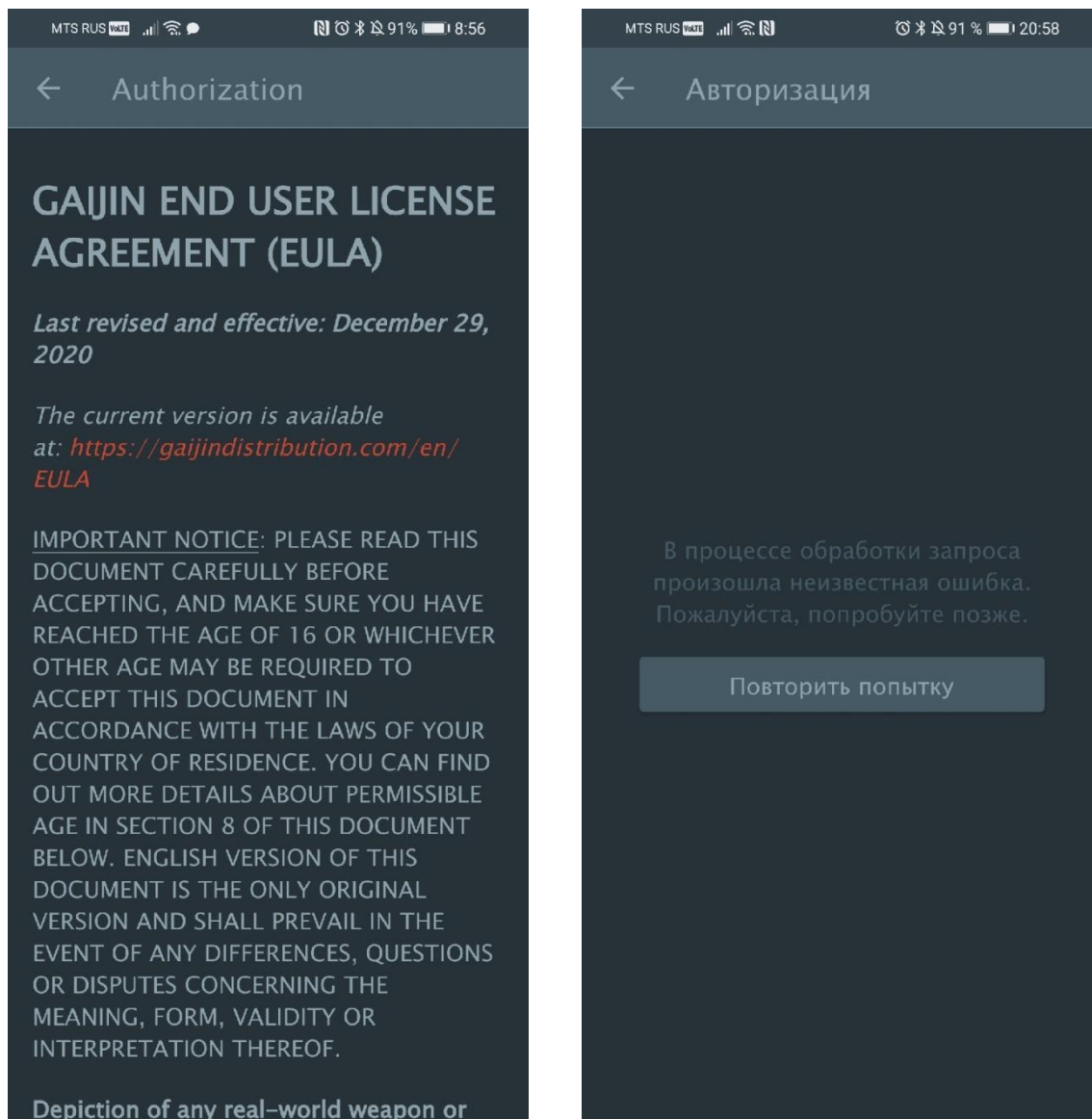


Рис.4, 5. Экран текста лицензионного соглашения.

2.2.5. StateEnterPassword

Окно ввода пароля. Пользователь должен ввести пароль, затем нажать кнопку «Войти» для продолжения авторизации. После нажатия кнопки будет

отправлен запрос на авторизацию на сервер. При этом к запросу может быть автоматически добавлен код TFA, если авторизуемый пользователь привязан в разделе безопасности приложения WTA или пользователь привязан в установленном на этом же устройстве приложении GaijinPass и к нему разрешен доступ.

Пользователь может разрешить доступ к приложению GaijinPass, нажав на кнопку «Разрешить» на экране ввода пароля. Эта процедура не требует получения дополнительных системных разрешений, фактически представляя собой установку разрешающего флага в настройки приложения. При необходимости пользователь может также отключить автоматическое получение кодов авторизации из GaijinPass, убрав флажок в разделе настроек.

Возможные переходы:

1. По событию «Назад» пользователь вернется к экрану `StateEnterLogin`.
2. По ссылке «Забыли пароль?» откроется вкладка в системном браузере со страницей поддержки.
3. Успешная авторизация: окончание процесса авторизации.
4. При ошибке получения идентификационных данных по причине отсутствия кода двухфакторной авторизации пользователь будет направлен на экран `StateEnterCode`.
5. При ошибке получения идентификационных данных, не связанной с кодом TFA, или при ошибке получения данных профиля, не связанной с баном или заморозкой, отобразится окно с сообщением об ошибке.
6. При ошибке получения профиля в связи с баном или заморозкой аккаунта отобразится соответствующее сообщение с ошибкой и откроется экран ввода логина `StateEnterLogin`.

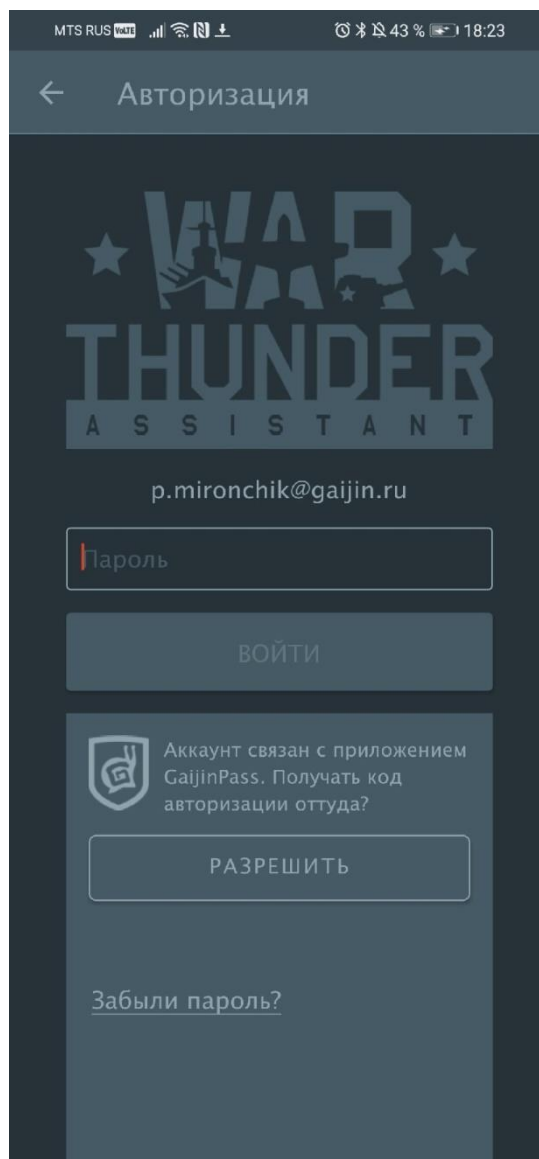


Рис.6. Экран ввода пароля.

2.2.6. StateEnterCode

Экран получения кода TFA. Этот экран показывается в том случае, если на этапе авторизации из состояния `StateEnterPassword` или при повторной авторизации была получена ошибка отсутствия кода TFA. Пользователь может ввести код в предложенное поле и по нажатию кнопки «Войти» запустить авторизацию, либо использовать автоматическое получение кода авторизации (см. раздел 1.2.1).

Раздел автоматического получения кода авторизации не показывается, если у пользователя нет устройств с привязанным приложением `GaijinPass` или `WTA`. Если автоматическое получение кода доступно, процесс его

получения начинается сразу с открытием экрана. Если по истечении 5 попыток код не был получен или произошла ошибка при какой-либо попытке, процесс прерывается и показывается кнопка «Запросить вход», по нажатию на которую процесс начинается заново.

При получении кода TFA, от пользователя или автоматически, отправляется запрос на авторизацию. Если авторизация не удастся по причине бана или заморозки аккаунта, открывается экран ввода логина с показом соответствующей ошибки, другие ошибки выводятся непосредственно над текущим экраном.

Пользователь может отметить флажок «Запомнить это устройство на 30 дней», чтобы запомнить это устройство и при авторизации в ближайший месяц это устройство не требовало ввода кода TFA.

По событию «Назад» или по нажатию кнопки «Отмена» открывается экран ввода пароля.

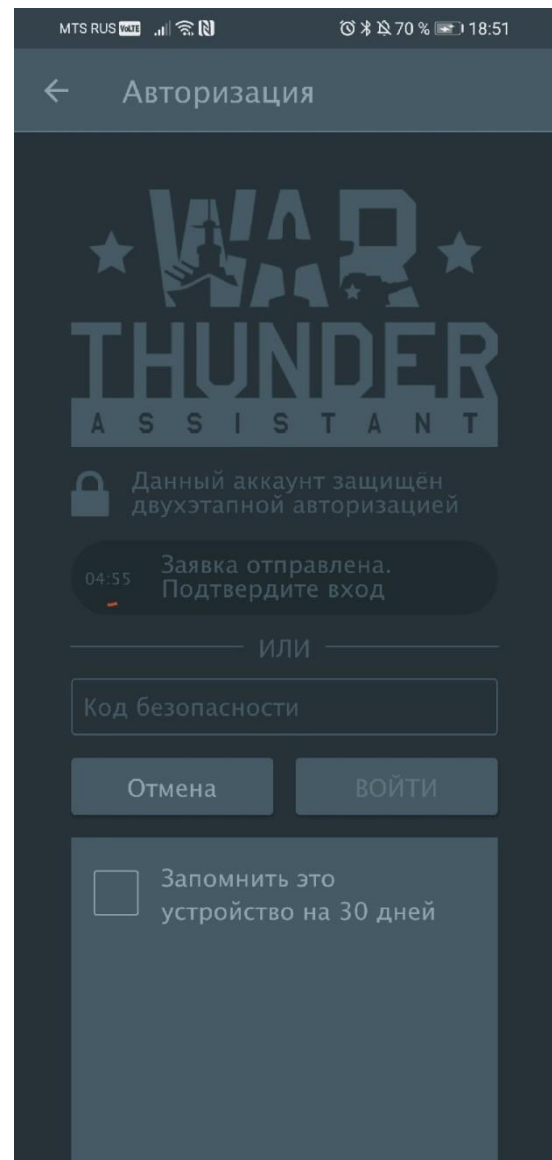
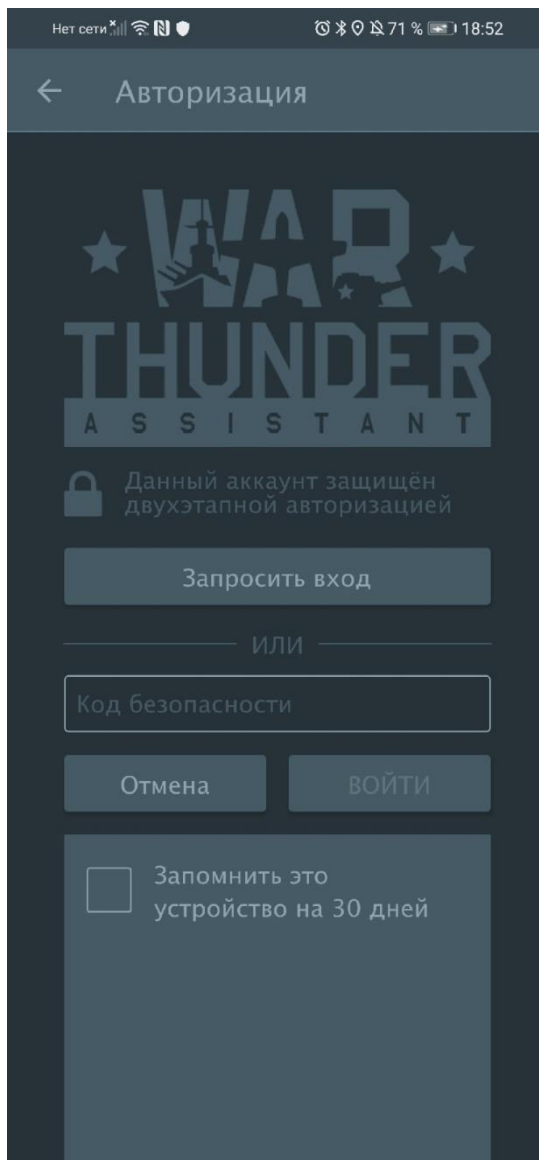


Рис.7, 8. Экран ввода кода TFA.

2.2.7. StateSteamFetchUrl

Состояние получения ссылки на авторизацию Steam OpenID. Сразу по старту состояния отправляется запрос на сервер авторизации для получения ссылки. Интерфейс окна представлен стандартным виджетом AppBar и индикатором загрузки в центре экрана. Возможные переходы:

1. При событии «Назад» открывается экран StateEnterLogin. Запрос получения ссылки при этом отменяется.
2. При возникновении ошибки в процессе получения ссылки пользователю выводится ошибка и открывается экран StateEnterLogin.

3. При успешном получении ссылки открывается состояние `SteamWebViewState`.

2.2.8. SteamWebViewState

Экран, где пользователь должен авторизоваться в Steam аккаунте.

Экран состоит из виджета `AppBar` и системного виджета `WebView` для отображения и загрузки web страницы. В `WebView` открывается страница по ссылке, полученной в состоянии `StateSteamFetchUrl`. Приложение не вмешивается в процесс авторизации на сайте, и перехватывает событие переадресации после завершения авторизации.

Пользователь может отменить авторизацию через событие «Назад» и вернуться в экран `StateEnterLogin`, либо завершить авторизацию на сайте.

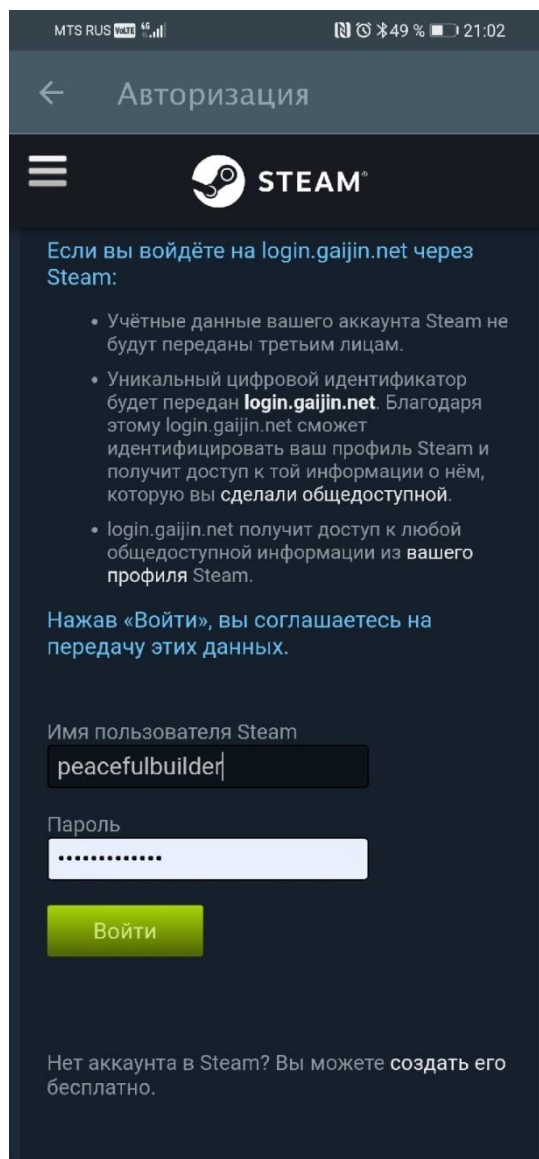


Рис. 9. Экран авторизации Steam OpenID.

2.2.9. StateSteamLogin

Авторизация на сервере приложения через данные Steam аккаунта. Из перехваченного в состоянии `SteamWebViewState` адреса получают параметры, которые отправляются с запросом авторизации на сервер приложения, ответом на который являются идентификационные данные аккаунта. Если идентификационные данные и данные профиля успешно получены, процесс авторизации завершается; в противном случае пользователь вернется на экран `StateEnterLogin`. Интерфейс аналогичен экрану `StateSteamFetchUrl`. Возможные события:

1. По событию «Назад» откроется экран `StateEnterLogin`.

2. При неудачном получении идентификационных данных пользователю отобразится ошибка с переходом на экран StateEnterLogin.
3. При неудачном получении данных профиля, включая бан и заморозку, отобразится ошибка с переходом на экран StateEnterLogin.
4. При успешном получении идентификационных данных и данных профиля процесс авторизации завершится.

ЗАКЛЮЧЕНИЕ

В ходе прохождения производственной практики был реализован сервис авторизации для приложения War Thunder Assistant. Архитектура сервиса основана на машине состояний, которые позволяют разделить обязанности по разным объектам и упростить структуру. Авторизация может быть проведена двумя способами: используя пару логин-пароль с кодом двухфакторной авторизации и через сервис Steam OpenID, открытый в встроенном браузере. Ошибки, возникающие в процессе авторизации, корректно обрабатываются и при необходимости выводятся пользователю.

При реализации задачи были использованы следующие технологии:

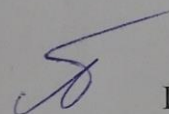
- Flutter - основной фреймворк;
- Dart - язык фреймворка Flutter;
- protobuf - сериализация данных профиля;
- JSON - сериализация идентификационных данных и основной формат данных в запросах;
- HTML;
- HTTPS;
- RSA - генерация ключей при авторизации;
- JWT - формат токена;
- OpenID - протокол авторизации в Steam аккаунте.

Отзыв руководителя

Во время летней практики студент Мирончик П.Д. занимался разработкой сервиса авторизации для приложения War Thunder Assistant на основе фреймворка flutter. Необходимо было реализовать и встроить в существующее приложение сервис с возможностью двухфакторной авторизации с использованием логина и пароля и авторизации через сервис Steam OpenID. Работа выполнена качественно, студент постоянно поддерживал связь с рабочей группой.

Рекомендуемая оценка: отлично.

Руководитель



Петросов С.Ю.