

Технически университет – София

Факултет “Компютърни системи и технологии”

Курсова Работа

ПО

„Програмиране за мобилни устройства.“

Георги Антонов Трапов, 121221185, 44Б група - Програмист Отговорник
Димитър Василев Иванов, 121221061, 44Б група - Програмист Отговорник
Мilan Анатолий Milанов, 121221101, 44Б група - Дизайнер Отговорник

София, 2024г.

Съдържание

Съдържание.....	2
1. Увод.....	6
2. Анализ на вече съществуващи решения.....	7
2.1 Технически университет - София “Е-Студент”.....	7
2.1.1. Какво представлява ?.....	7
2.1.2. Предимства.....	7
2.1.3. Недостатъци.....	7
2.1.4. Изглед от системата.....	8
2.2. Софийски Университет “Св. Климент Охридски” - Информационна система	8
2.2.1. Какво представлява ?.....	8
2.2.2. Предимства.....	9
2.2.3. Недостатъци.....	9
2.2.4. Изглед от системата.....	10
2.3. Е-студент Великотърновски Университет “Св. Св. Кирил и Методий”.....	10
2.2.3. Предимства.....	10
2.2.4. Недостатъци.....	11
3. Изводи за съществуващите решения.....	12
2.2.4. Изглед от системата.....	12
4. Проектиране на проекта.....	13
4.1. Функционални изисквания.....	13
4.2. Нефункционални изисквания.....	14
5. Обхват.....	15
6. Use case диаграми.....	15
6.1. Диаграма на студент.....	15
6.2. Диаграма на преподавател.....	16
6.3. Диаграма канцелария.....	16
7. Диаграма на компонентите.....	17
8. Съхранение на данни.....	18
9. Deployment диаграма.....	19
10. Flow chart.....	20
11. Хардуер.....	21
12. Използвани технологии.....	22
13. Използвани библиотеки.....	23

13.1. Фронтенд библиотеки.....	23
13.1.1. expo/metro-runtime.....	23
13.1.2. react-native-async-storage/async-storage.....	23
13.1.3. react-navigation/drawer.....	23
13.1.4. react-navigation/native.....	23
13.1.5. react-navigation/native-stack.....	23
13.1.6. types/react-native.....	23
13.1.7. axios.....	23
13.1.8. expo.....	23
13.1.9. expo-build-properties.....	24
13.1.10. expo-checkbox.....	24
13.1.11. expo-constants.....	24
13.1.12. expo-device.....	24
13.1.13. expo-local-authentication.....	24
13.1.14. expo-notifications.....	24
13.1.15. expo-status-bar.....	24
13.1.16. react.....	24
13.1.17. react-dom.....	24
13.1.18. react-native.....	24
13.1.19. react-native-big-calendar.....	25
13.1.20. react-native-gesture-handler.....	25
13.1.21. react-native-paper.....	25
13.1.22. react-native-push-notification.....	25
13.1.23. react-native-reanimated.....	25
13.1.24. react-native-safe-area-context.....	25
13.1.25. react-native-screens.....	25
13.1.26. react-native-web.....	25
13.1.27. react-native-picker.....	25
13.1.28. react-native-community/datetimepicker.....	25
13.2. Бекенд библиотеки.....	26
13.2.1. axios.....	26
13.2.2. cross-env.....	26
13.2.3. laravel-mix.....	26
13.2.4. lodash.....	26
13.2.5. resolve-url-loader.....	26
14. Бекенд.....	26
14.1. Модели.....	26
14.1.1 .Discipline.....	26
14.1.2. Dorm.....	27

14.1.3. Event.....	28
14.1.4. Faculty.....	28
14.1.5. Grade.....	29
14.1.5. GrantRequest.....	30
14.1.6. Group.....	31
14.1.7. Insurance.....	31
14.1.8. Specialty.....	32
14.1.9. Sport.....	33
14.1.10. Stream.....	33
14.1.11. StudentSport.....	34
14.1.12. StudentSport.....	35
14.2. Контролери.....	36
14.2.1. DisciplineController.....	36
14.2.2. EventController.....	37
14.2.3. InsuranceController.....	37
14.2.4. InsuranceController.....	37
14.2.5. StudentController.....	38
14.2.6. TeacherController.....	41
14.2.7. UserContoller.....	43
14.3. Сървиси.....	45
14.3.1.....	45
14.4. Мидълуер.....	46
14.4.1. Authenticate.....	46
14.4.2. RedirectIfAuthenticated.....	46
15. Фронтенд.....	48
15.1. Assets.....	48
15.2. Компоненти.....	48
15.2.1. Form.....	48
15.2.2. Auth Navigator.....	50
15.3. Контекст.....	54
15.3.1. AuthContext.....	54
15.3.2. Push Context.....	55
15.4. Екрани.....	55
15.4.1. SudentCalendarScreen.....	55
15.4.2. SudentDormScreen.....	56
15.4.3. SudentEventScreen.....	59
15.4.4. SudentGradeScreen.....	61

15.4.5. StudentGrantScreen.....	63
15.4.6. TeacherInsuranceScreen.....	66
15.4.7. TeacherSporScreen.....	68
15.4.8. TeacherGradeDisciplineScreen.....	70
15.4.9. TeacherGradeHomeScreen.....	72
15.4.10. TeacherGradeScreen.....	73
15.4.11. LoginScreen.....	74
15.4.12. LogoutScreen.....	78
15.4.13. TeacherGradeScreen.....	78
16. Потребителски интерфейс и ръководство.....	81
16.1. Цветове и дизайн.....	81
16.2. Вход в приложението.....	81
16.3. Информация за студена.....	83
16.4. Оценки на студента.....	85
16.5. Календар на студента.....	86
16.6. Събития на студента.....	87
16.7. Здравно осигуряване на студента.....	88
16.8. Стипендии, Общежития, Спорт на Студента.....	89
16.9. Информация за преподавател.....	90
16.10. Екран за оценки - преподавател.....	91
16.11. Екран спортове - преподавател.....	92
16.12. Информация за служител.....	93
16.13. Календар на служителя.....	94
16.14. Събития на служителя.....	95
16.15. Добави събития.....	96
17. Бъдещо развитие.....	98
17.1 . Интеграция със системи за управление на учебни материали.....	98
17.2. Подобрени социални функции.....	98
17.3. Система за стажове и работни места за студентите.....	98
17.4 Интерактивни карти на кампуса.....	98
17.5. Нощен режим.....	98
18. Заключение.....	99
19. Използвана литература.....	100
19.1. OPNsense.....	100
19.2. React Native.....	100
19.3. Laravel.....	100

1. Увод

Мобилните устройства са неотделима част от ежедневието на студентите. Затова представяме нашето мобилно приложение за студентска система, което цели да улесни и подобри учебния процес. С нашето приложение студентите ще имат лесен достъп до информацията за тяхното образование - от оценките и програмата до актуални съобщения от университета. Нашата цел е да осигурим на студентите удобство и функционалност на всяко ниво на техния образователен път, като им предоставим инструмент, който да ги подкрепя в тяхната академична и личностна растеж.

2. Анализ на вече съществуващи решения

2.1 Технически университет - София "Е-Студент"

2.1.1. Какво представлява ?

"Е-студент" е онлайн платформата на Технически университет - София, която обхваща всички аспекти от учебния процес за студентите. Тя включва информация като оценки, академични заверки, информация за стипендии, настаняване в общежития и други.

2.1.2. Предимства

Всичката информация в нея е обстойна и достъпна лесно чрез компютър.

2.1.3. Недостатъци

Липсва приложение за мобилни устройства.

Уеб страницата не разполага с добре адаптивен дизайн, което я прави трудна за използване на телефон.

Входът в системата изисква потребителско име и парола при всяко отваряне.

Не се получават нотификации при поставяне на нова оценка на студента

2.1.4. Изглед от системата

Фиг. 2.1.4.1.

Фиг. 2.1.4.2.

Фиг. 2.1.4.3.

Технически университет - София

"Е-Студент"

Фак. № []
ЕГН/ЛНЧ []
Текст от изображението

Вход

© ТУ - София 2013 - 2024.

Технически университет - София

"Е-Студент"

Ако не виждате някоя дисциплина, има вероятност да не е зададена нейната форма на контрол в учебния план. Съобщете за проблема във Вашата студентска канцелария.

Дисциплина (форма на контрол)	Лекции	Лаб. упр.	Сем. упр.
1 сем.			
Математика I (Изпит), оценка: Мн. добър (5) (редовна), по протокол № 663889 от 31.01.2022, 12:20:47	проф. д-р [] 27.12.2021, 22:57:10	проф. д-р [] 27.12.2021, 22:57:10	
Физика (Изпит), оценка: Среден (3) (редовна), по протокол № 663892 от 27.01.2022, 20:20:46	проф. д-р [] 16.12.2021, 18:54:46	ас. [] 23.12.2021, 09:49:14	гл. ас. д-р [] 29.12.2021, 17:18:37
Въведение в програмирането (Изпит), оценка: Среден (3) (редовна), по протокол № 663895 от 23.01.2022, 21:50:46	проф. д-р [] 04.01.2022, 09:22:51	ас. [] 22.12.2021, 12:00:48	гл. ас. д-р [] 22.12.2021, 12:52:48
Основи на инженерното	проф. д-р []	проф. д-р []	проф. д-р []

Информация
Здравно осигуряване
Заверки и оценки
Спорт
Стипендии
Общежития
Плащания
Идентификация
История на влизанията
Изход

достъп е вашето ЕГН

Хронология на студентското състояние

Курс	Уч. год.	Зав. сем.	Състояние	Причина	Зап. №	Зап. д.
1	2021/2022		Действащ	Прием - ТУ	1755	09.07.2

2.2. Софийски Университет “Св. Климент Охридски” - Информационна система

2.2.1. Какво представлява ?

Онлайн платформата на Софийския университет "Св. Климент Охридски" представлява информационна система, която обхваща всички важни аспекти от учебния процес за студентите. Тази система предоставя информация за оценки, академични заверки, налични стипендии, процеса на настаняване в общежития и други.

2.2.2. Предимства

Всичката информация в нея е обстойна и достъпна лесно през уеб страницата.

2.2.3. Недостатъци

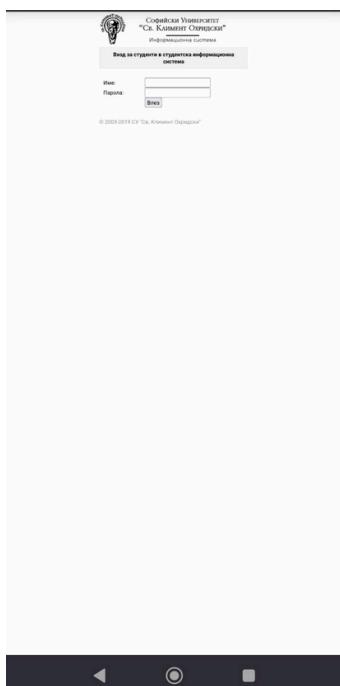
Няма налично приложение за мобилни устройства.

Уеб страницата не разполага с адаптивен дизайн, който я прави трудна за използване на телефон.

Входът в системата изисква потребителско име и парола при всяко отваряне.

Не се получават известия при добавяне на нова оценка на студентите.

2.2.4. Изглед от системата



Фиг. 2.2.4.1



Фиг. 2.2.4.2



Фиг. 2.2.4.3

2.3. Е-студент Великотърновски Университет "Св. Св. Кирил и Методий"

Е-студент Великотърновски Университет "Св. Св. Кирил и Методий" е онлайн платформа, която цели да улесни и подобри образователния процес за студентите. Тази платформа предоставя цялостна информационна система, която обхваща всички ключови аспекти от учебния живот на студентите.

2.2.3. Предимства

Лесен достъп до информация: Бърз и удобен достъп до академични резултати и информация за студентите.

Удобство и ефективност: Централизирана система за управление на учебния процес, което улеснява организацията и планирането.

Подобрена комуникация: Възможност за лесна комуникация между студенти и преподаватели.

Управление на събития и срокове: Календар със събития и срокове, което помага за по-добро управление на времето.

Информация за стипендии: Лесен достъп до информация за наличните стипендии и условията за кандидатстване за тях.

Управление на общежития: Цялостно управление на процеса на настаняване в общежития и общежитийни услуги.

2.2.4. Недостатъци

Липсва мобилно приложение, което ограничава достъпа до платформата чрез мобилни устройства.

Уеб страницата не е адаптивна и не се приспособява към различните размери на екрана на мобилните телефони, което затруднява използването на платформата.

Входът в системата изисква потребителско име и парола при всеки вход, което може да бъде неудобно за потребителите.

Липса на известия за студентите при добавяне на нови оценки, което може да доведе до липса на своевременна информация за тяхните академични постижения.

3. Изводи за съществуващите решения

Има нужда от университетска система, която е лесно достъпна и през мобилни устройства, за да отговаря на нуждите студентите, които предпочитат да използват телефоните си за достъп до информация. Такава платформа би предоставила на студентите удобство и гъвкавост при проверка на оценки и важна информация.

2.2.4. Изглед от системата

Фиг. 2.2.4.1



Фиг 2.2.4.2.

График на занятията - общ						
Семестър / Занятие	Филологически факултет	Ден	Час	Седмица	Трупа	Лекции
Понеделник	10.15-12.00	всеки		3/21	Ректорат	Лекции
Понеделник	12.15-14.00	всеки	1	207	Ректорат	Управление
Понеделник	12.15-14.00	всеки	2	440	Ректорат	Управление
Понеделник	14.15-16.00	всеки	1	508	Ректорат	Управление
Понеделник	14.15-16.00	всеки	2	207	Ректорат	Управление
Понеделник	16.15-18.00	1,3	1	207	Ректорат	Управление
Понеделник	16.15-18.00	2,4	2	207	Ректорат	Управление
Вторник	14.15-16.00	2,4	1	207	Ректорат	Управление
Вторник	14.15-16.00	1,3	1	208	Ректорат	Управление
Вторник	14.15-16.00	2,4	2	208	Ректорат	Управление
Вторник	16.15-18.00	1,3	1	1	Спорти	Управление
Вторник	16.15-18.00	2,4	2	207	Ректорат	Управление
Вторник	18.00-18.15	1,3	2	1	Спорти	Управление

Резултати от изпитите								
	Сем.	Дисциплина	Л	У	Оценка	Кр.	Преподавател	Протокол
Ръководство за работа РЗ	1	Английски език I част	0	105	мн. добър	5	16	призната с протокол P504
Персонални данни	1	Компютърна текстообработка	15	15	неположен	-	0	гл.ас. д-р Димо Милев Милев i115620
Изучавани дисциплини	1	Увод в езикознанието	30	15	мн. добър	5	4	призната с протокол P504
Моби / Завършения	1	Фонетика и фонология на английския език	30	15	мн. добър	5	6	призната с протокол P504
Индивидуални протоколи	2	Английска литература XIX век	30	15	неположен	-	0	док. д-р Ярмила Николова Даскаловова i115621
Заповеди	2	Английски език II част	0	90	добър	4	12	призната с протокол P504
Заверени семестри	2	Български език и стил	0	30	мн. добър	5	4	призната с протокол P504
Платени такси	2	История и култура на Великобритания и Ирландия	30	0	добър	4	4	призната с протокол P504
Държавни изпити	2	Латински език	0	30	мн. добър	5	2	призната с протокол P504
График на занятията	2	Литературознание	30	0	неположен	-	0	
Филтриран график на занятия	2	Физическо възпитание и спорт	0	30	отличен	6	2	призната с протокол P504
График на изпитите	3	Английска литература (XX-XXI век)	30	15	мн. добър	5	4	призната с протокол P504
Молба за обаждение								
Анкетни проучвания								

Фиг 2.2.4.3

4. Проектиране на проекта

4.1. Функционални изисквания

Таб 4.1

№	Изискване	Приоритет
1	Приложението трябва да позволява влизане факултетен номер и егн	Критичен
2	Приложението трябва да позволява влизане с пръстов отпечатък или лицево разпознаване	Нормален
3	Приложението да позволява на потребител влезнал като студент да вижда всички семестри, всички дисциплини и всички оценки към тях	Критичен
4	Приложението да позволява потребителя да вижда средния си успех	Нормален
5	Приложението да има календа, на който да се виждат предстоящите му часове	Висок
6	Приложението да позволява на студента да вижда ОКС, факултетен номер, специалност, група, записан семестър и имейл към университета	Критичен
7	Приложението да изпраща нотификации при добавяне на нова оценка на студента	Висок
8	Приложението да изпраща нотификации при ново съобщение от университета	Нормален
9	Приложението да показва информация за преподавател	Нисък

4.2. Нефункционални изисквания

Таб 4.2

№	Изискване	Приоритет
1	Системата да е устойчива и да и да може да обслужва поне 300 студента	Висок
2	Дизайн, удобен за използване от мобилно устройство	Висок
3	Системата да не зарежда повече от 1 секунда при наличието на добър интернет	Нормален
4	Системата да използва трислойна архитектура	Критичен
5	Системата да използва релационна база от данни	Висок
6	Студента да получава нотификации дори когато приложението не е на фон	Нормален
7	Приложението да е изцяло на български език	Нормален

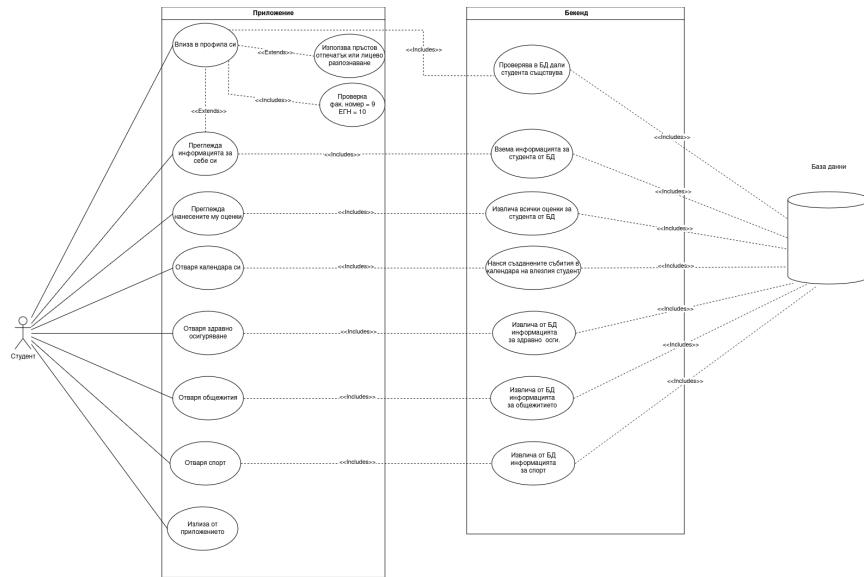
5. Обхват

Приложението за университетска система има за цел да обхване всички студенти от Технически Университет - София. Ще предостави удобен начин за студентите да проследяват своите занятия, оценки и други актуални събития, свързани с учебния процес. Освен това, то може да бъде полезно и за университетската администрация, като предоставя по-ефективни инструменти за управление на информацията и комуникацията със студентите.

6. Use case диаграмми

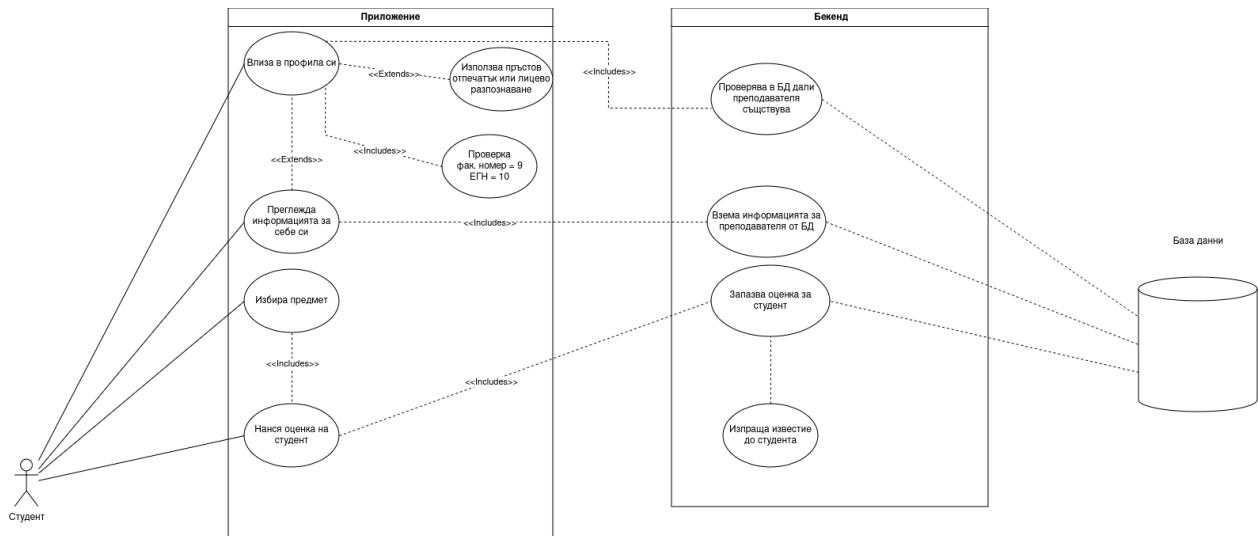
6.1. Диаграма на студент

Фиг 6.1



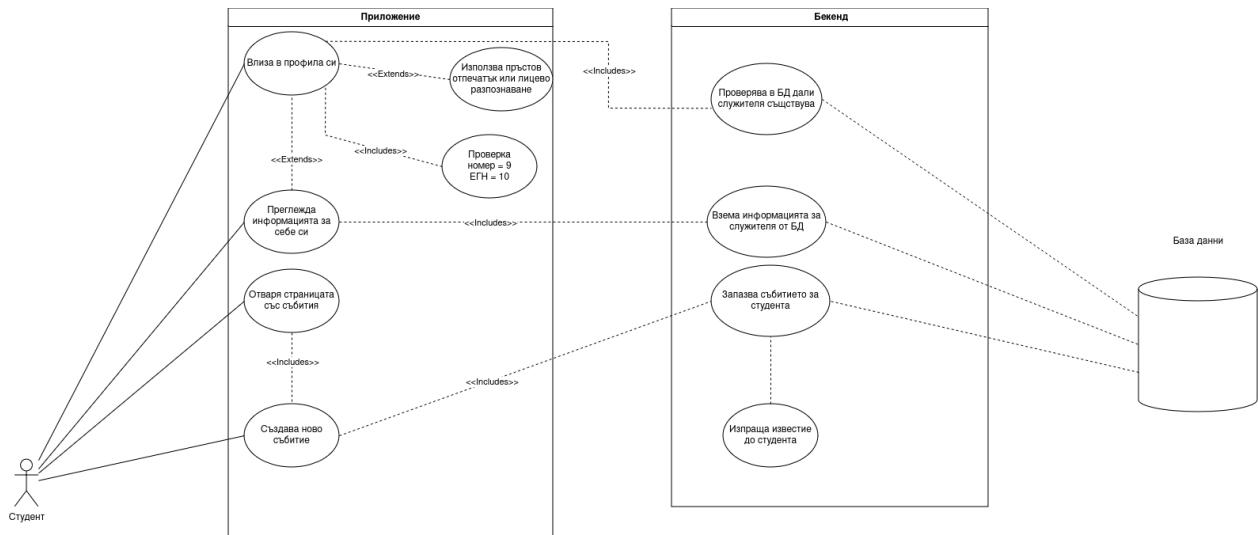
6.2. Диаграма на преподавател

Фиг 6.2



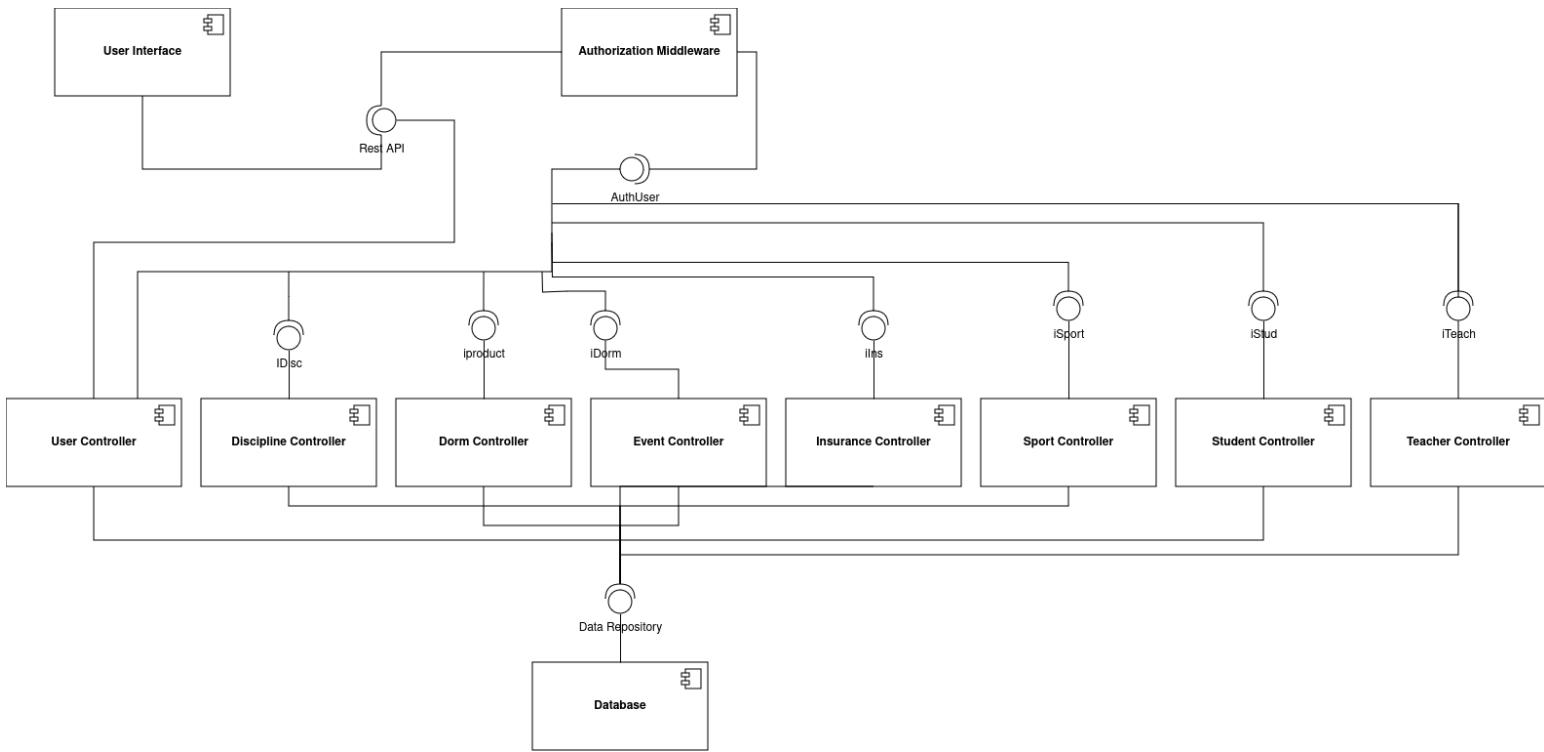
6.3. Диаграма канцелария

Фиг 6.3



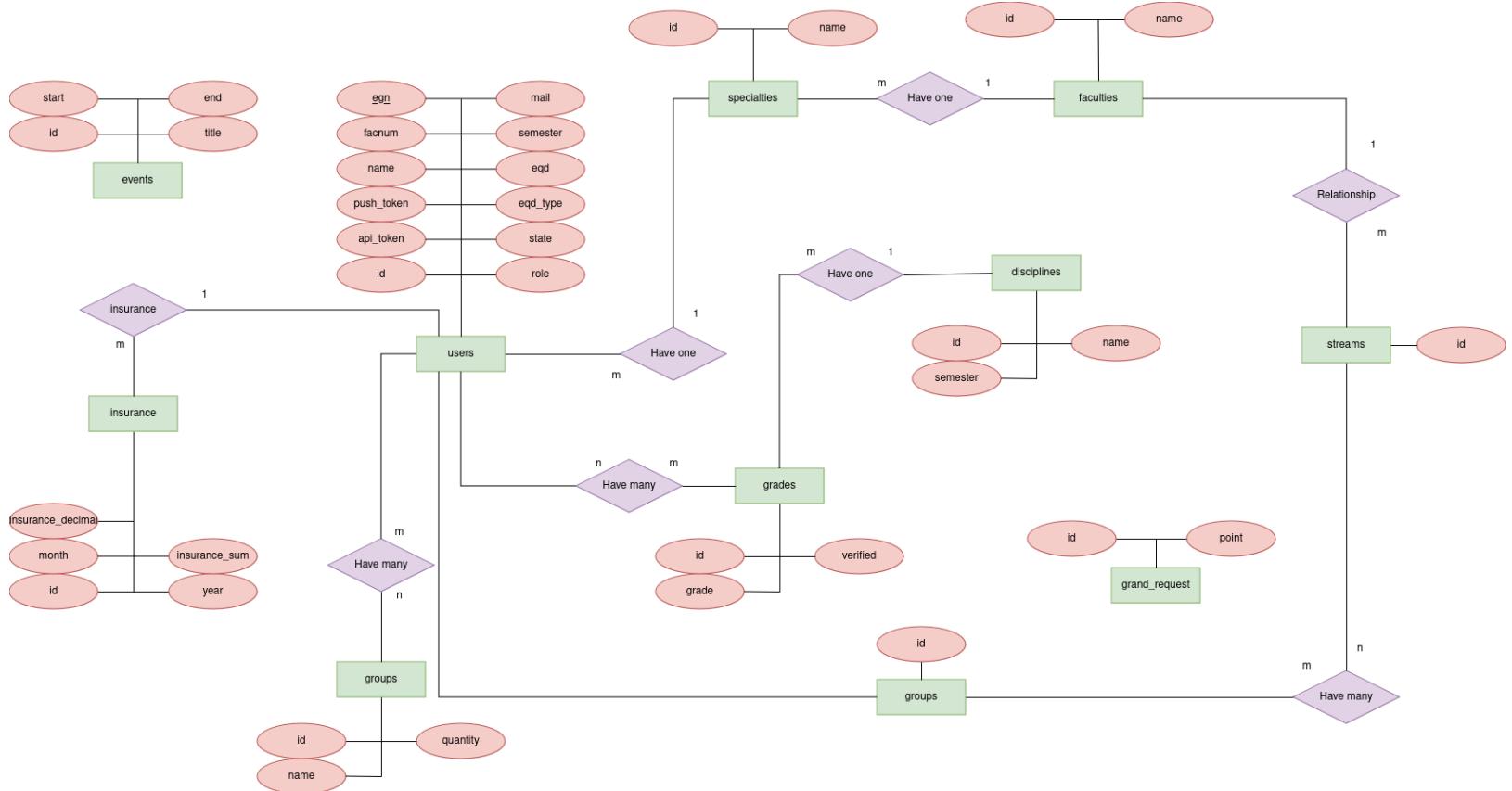
7. Диаграма на компонентите

ФИГ 7.1



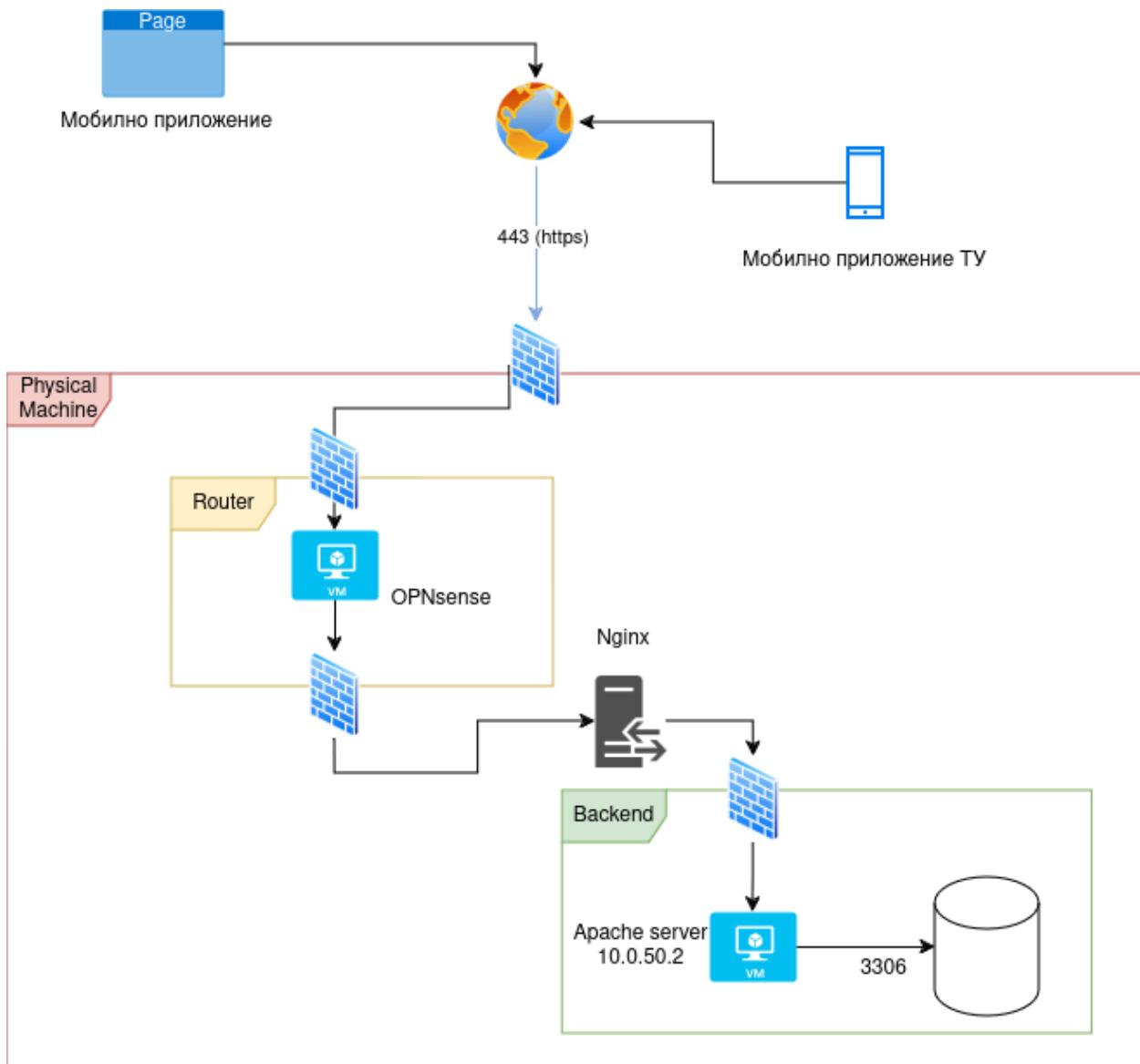
8. Съхранение на данни

Фиг. 8



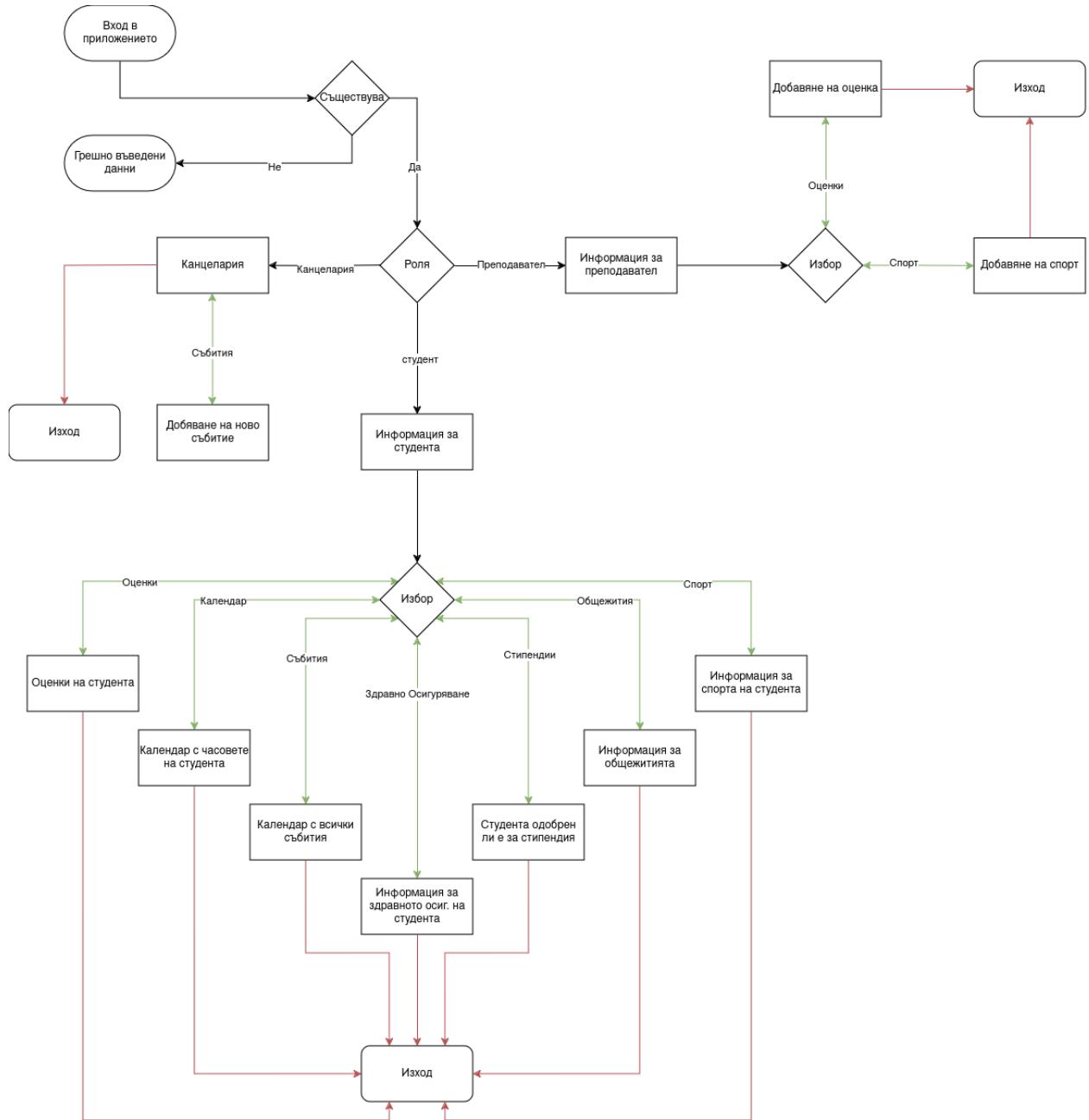
9. Deployment диаграмма

Фиг 9.1



10. Flow chart

Фиг. 10



11. Хардуер

Приложението ще бъде хоствано на отделна машина, която е конфигурирана с OPNsense за защита и управление на мрежовия трафик. Бекендът, базата данни и рутера ще бъдат разположени на отделни виртуални машини, което осигурява по-голяма гъвкавост и сигурност в управлението на компонентите на инфраструктурата. Този подход позволява по-добро разграничаване на ролите и по-ефективно управление на ресурсите.

Фиг 11.1



Фиг 11.2



Фиг. 11.3



Фиг. 11.4

12. Използвани технологии

Разработали сме иновативно приложение, което обхваща както мобилна, така и уеб част, използвайки няколко ключови технологии, за да осигурим най-доброто потребителско изживяване и функционалност.

За мобилната част на приложението сме избрали **React Native** - технология, която ни позволява да създадем едно приложение, което работи както на **iOS**, така и на **Android**.

За уеб частта, използваме **PHP Laravel** - фреймуърк с висока производителност и елегантен синтаксис, който ни помага да създадем сигурни и скалабилни уеб приложения. **Laravel** предоставя гъвкави инструменти за управление на рутинните операции като маршрутизация, автентикация и взаимодействие с база данни.

За осигуряване на сигурността на приложението използваме **OpenSense**, който ни позволява да контролираме и мониторираме достъпа до системата, като същевременно откриваме и реагираме на потенциални заплахи и атаки.

Относно комуникацията с потребителите, използваме **Firebase** за нотификации. **Firebase** предоставя гъвкави инструменти за изпращане на съобщения в реално време до потребителите.

За съхранение на данни използваме **MySQL** - един от най-популярните релационни управляващи системи, който ни осигурява надеждно и ефективно съхранение на информацията.

Накрая, за управление на уеб сървъри използваме както **Nginx** и **Apache**. Те ни предоставят висока производителност, сигурност и надеждност.

13. Използвани библиотеки

13.1. Фронтенд библиотеки

13.1.1. expo/metro-runtime

Помощна библиотека за изграждане и пакетиране на приложения в Expo.

13.1.2. react-native-async-storage/async-storage

Библиотека за асинхронно съхранение на данни на мобилни устройства, използвана за съхранение на данни в приложения React Native.

13.1.3. react-navigation/drawer

Навигационна библиотека за React Native, която позволява създаването на изтеглящи се панели (drawer navigation) в приложениета.

13.1.4. react-navigation/native

Библиотека за навигация за React Native, която предоставя инструменти за навигация между различни екрани в приложението.

13.1.5. react-navigation/native-stack

Стекова навигационна библиотека за React Native, която управлява навигацията между екрани в стека на приложението.

13.1.6. types/react-native

Библиотека с TypeScript декларации за React Native, които предоставят типова информация за използваните модули.

13.1.7. axios

Библиотека за извършване на HTTP заявки към уеб сървъри.

13.1.8. expo

Инструментариум за разработка на приложения, които използват React Native и се изпълняват на мобилни устройства.

13.1.9. expo-build-properties

Помощна библиотека за Expo, която предоставя информация за собствеността на сградата.

13.1.10. expo-checkbox

Компонент за отметка (checkbox) за приложения, разработени с Expo.

13.1.11. expo-constants

Библиотека, която предоставя константи и информация за устройството, на което се изпълнява Expo приложението.

13.1.12. expo-device

Библиотека за Expo, която предоставя информация за устройството, на което се изпълнява приложението.

13.1.13. expo-local-authentication

Библиотека за автентикация на местно ниво за приложения, изпълняващи се в Expo.

13.1.14. expo-notifications

Библиотека за управление на известията в приложенията, разработени с Expo.

13.1.15. expo-status-bar

Компонент за статусна лента (status bar) за приложения, разработени с Expo.

13.1.16. react

Библиотека за разработка на интерфейси (UI) в JavaScript, която се използва за създаване на потребителски интерфейси в React Native.

13.1.17. react-dom

Помощна библиотека за React, която позволява рендерирането на компоненти в DOM (Document Object Model) на уеб приложениета.

13.1.18. react-native

Фреймуърк за създаване на мобилни приложения с помощта на JavaScript и React.

13.1.19. react-native-big-calendar

Компонент за календар с голям формат за мобилни приложения, разработени с React Native.

13.1.20. react-native-gesture-handler

Библиотека за обработка на жестове (gestures) в мобилни приложения, разработени с React Native.

13.1.21. react-native-paper

Библиотека за дизайн на компоненти за мобилни приложения, базирани на Material Design.

13.1.22. react-native-push-notification

Библиотека за изпращане на известия (push notifications) в мобилни приложения, разработени с React Native.

13.1.23. react-native-reanimated

Библиотека за анимация в реално време в мобилни приложения, разработени с React Native.

13.1.24. react-native-safe-area-context

Библиотека за управление на безопасните области на екрана (safe area) в мобилни приложения, разработени с React Native.

13.1.25. react-native-screens

Библиотека за управление на экраните в мобилни приложения, разработени с React Native.

13.1.26. react-native-web

Библиотека, която позволява използването на React Native компоненти в уеб приложения.

13.1.27. react-native-picker

Компонент за избор (picker) в мобилни приложения, разработени с React Native.

13.1.28. react-native-community/datetimepicker

Компонент за избор на дата и време в мобилни приложения, разработени с React Native.

13.2. Бекенд библиотеки

13.2.1. axios

Библиотека за извършване на HTTP заявки от страна на клиента към сървър.

13.2.2. cross-env

Позволява дефинирането на променливи за околната среда, които се използват във всички операционни системи.

13.2.3. laravel-mix

Мощен инструмент за сборка на ресурси, който се използва за компилиране на JavaScript, Sass и други ресурси в проекта.

13.2.4. lodash

Популярна библиотека за работа с масиви, обекти и други данни, която предоставя удобни и ефективни методи за манипулация на данни.

13.2.5. resolve-url-loader

Webpack loader, който резултира в правилно разрешение на относителни пътища във Sass и CSS файлове.

14. Бекенд

14.1. Модели

14.1.1 .Discipline

Този PHP код представлява модел в Laravel приложение за дисциплини. Моделът се казва **Discipline** и има полета **id**, **name**, **semester** и **specialty_id**. Полето **id** е защитено, докато останалите полета могат да бъдат масово присвоени. Моделът има връзка много към едно с модела **Specialty**, което указва, че всяка дисциплина принадлежи на определена специалност.

```
use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Discipline extends Model

{

    protected $guarded = [];

    protected $fillable = ["id", "name", "semester", "specialty_id"];



    public function specialty()

    {

        return $this->belongsTo(Specialty::class);

    }

    use HasFactory;
}
```

14.1.2. Dorm

Този PHP код представлява модел в **Laravel** приложение за общежития. Моделът се казва **Dorm** и има следните характеристики:

- **\$guarded = [];** позволява масово присвояване на всички полета.
- Методът **student()** указва, че всяко общежитие принадлежи на един потребител, дефиниран чрез връзка много към едно (many-to-one).
- **use HasFactory;** - използва фабриката за създаване на записи в базата данни.

```
class Dorm extends Model
{
    protected $guarded = [];

    public function student()
    {
        return $this->belongsTo(User::class);
    }

    use HasFactory;
}
```

14.1.3. Event

Този PHP код представлява модел в **Laravel** приложение за събития

```
class Event extends Model
{
    protected $guarded = [];

    use HasFactory;
}
```

14.1.4. Faculty

Този PHP код представлява модел в Laravel за факултети. Моделът е наименуван **Faculty**.

```
class Faculty extends Model

{
    protected $guarded = [];

    protected $fillable = ["id", "name"];

    public function specialties()
    {
        return $this->hasMany(Specialty::class);
    }

    public function streams()
    {
        return $this->hasMany(Stream::class);
    }

    use HasFactory;
}
```

14.1.5. Grade

Този PHP код представлява модел в Laravel за оценки (**Grades**).

```
class Grade extends Model
{
    protected $guarded = [];
```

```

protected $fillable = ["id", "grade", "verified", "student_id",
"teacher_id", "discipline_id"];


public function discipline()
{
    return $this->belongsTo(Discipline::class);
}

public function teacher()
{
    return $this->belongsTo(User::class);
}

public function student()
{
    return $this->belongsTo(User::class);
}

use HasFactory;
}

```

14.1.5. GrantRequest

Този PHP код представлява модел в Laravel за заявки за стипендия (**Grant Requests**)

Методът **student()** дефинира връзката на модела с модела **User** чрез метода **belongsTo()**.

```

class GrantRequest extends Model
{

```

```
protected $guarded = [];  
  
public function student()  
{  
  
    return $this->belongsTo(User::class);  
  
}  
  
use HasFactory;
```

14.1.6. Group

Този PHP код представлява модел в Laravel за групи (**Groups**).

Методите **specialty()** и **stream()** дефинират връзките на модела с моделите **Specialty** и **Stream** чрез метода **belongsTo()**.

```
class Group extends Model  
  
{  
  
    protected $guarded = [];  
  
    protected $fillable = ["id", "specialty_id", "stream_id",  
"specialty_id"];  
  
  
    public function specialty()  
    {  
  
        return $this->belongsTo(Specialty::class);  
  
    }  
  
  
    public function stream()  
    {  
  
        return $this->belongsTo(Stream::class);  
  
    }  

```

```
use HasFactory; }
```

14.1.7. Insurance

Този PHP код представлява модел в **Laravel** за застраховки (**Insurances**). Моделът се казва **Insurance**

Методът **student()** дефинира връзката на модела с модела **User**

```
class Insurance extends Model
{
    protected $guarded = [];

    public function student()
    {
        return $this->belongsTo(User::class);
    }

    use HasFactory;
}
```

14.1.8. Specialty

Този PHP код представлява модел в **Laravel** за специалности (**Specialties**)

Методът **disciplines()** дефинира връзката на модела с модела **Discipline**

Методът **faculty()** дефинира връзката на модела с модела **Faculty**

```
class Specialty extends Model
{
    protected $guarded = [];

    protected $fillable = ["id", "name", "faculty_id"];

    public function disciplines()
    {
```

```
    return $this->hasMany(Discipline::class);  
}  
  
public function faculty()  
{  
    return $this->belongsTo(Faculty::class);  
}  
use HasFactory;
```

14.1.9. Sport

Този PHP код представлява модел в Laravel за спортове (**Sports**).

Методът **teacher()** дефинира връзката на модела с модела **User**

```
class Sport extends Model  
{  
    protected $guarded = [];  
    public function teacher()  
{  
        return $this->belongsTo(User::class);  
    }  
use HasFactory;
```

14.1.10. Stream

Този PHP код представлява модел в Laravel за потоци (**Streams**)

Методът **groups()** дефинира връзката на модела с модела **Group**

Методът **faculty()** дефинира връзката на модела с модела **Faculty**

```
class Stream extends Model
```

```
{  
  
protected $guarded = [];  
  
protected $fillable = ["id", "faculty_id", "faculty_id"];  
  
  
public function groups()  
{  
  
    return $this->hasMany(Group::class);  
  
}  
  
  
public function faculty()  
{  
  
    return $this->belongsTo(Faculty::class);  
  
}  
  
  
use HasFactory;}  

```

14.1.11. StudentSport

Този PHP код представлява модел в **Laravel** за студентски спортни дейности (**StudentSport**).

Методът **student()** дефинира връзката на модела с модела **User**

Методът **sport()** дефинира връзката на модела с модела **Sport**

```
class StudentSport extends Model  
{  
  
protected $guarded = [];  
  
protected $table = 'student_sport';  

```

```

public function student()

{
    return $this->hasOne(User::class);
}

public function sport()

{
    return $this->hasOne(Sport::class);
}

use HasFactory;

```

14.1.12. StudentSport

Този PHP код представлява модел в **Laravel** за потребители (**Users**).

Методът **student_grades()** и **teacher_grades()** дефинират връзката на модела с модела **Grade**

Методът **dorm()** и **group()** дефинират връзката на модела с моделите **Dorm**

Методът **sport()** дефинира много-много връзката между потребителите и спортовете.

```

class User extends Model

{
    protected $guarded = [];

    protected $fillable = ["id", "api_token", "name", "facnum", "egn",
"mail", "eqd", "eqd_type", "state", "group_id"];


    public function student_grades()
    {
        return $this->hasMany(Grade::class, "student_id");
    }
}

```

```
}

public function teacher_grades()
{
    return $this->hasMany(Grade::class, "teacher_id");
}

public function dorm()
{
    return $this->hasOne(Dorm::class);
}

public function group()
{
    return $this->hasOne(Group::class);
}

public function sport()
{
    return $this->belongsToMany(Sport::class, StudentSport::class,
"student_id");
}

use HasFactory;
}
```

14.2. Контролери

14.2.1. DisciplineController

Този код представлява метод disciplines на контролера **DisciplineController**, който връща всички записи от таблицата за дисциплини в базата данни.

```
class DisciplineController extends Controller

{
    public function disciplines(Request $request)
    {
        return Discipline::all();}}
```

14.2.2. EventController

Този код представлява метод на контролера **EventController**, наречен **events**, който връща всички събития от базата данни.

```
class EventController extends Controller

{
    public function events(Request $request)
    {
        return Event::all();}}
```

14.2.3. InsuranceController

Този код представлява метод на контролера **InsuranceController**, наречен **insurances**, който връща всички записи за застраховки от базата данни.

```
class InsuranceController extends Controller

{
    public function insurances(Request $request)
```

```
{  
    return Insurance::all(); } }
```

14.2.4. InsuranceController

Този код представлява метод на контролера **SportController**, наречен **sports**, който връща всички записи за спорт от базата данни.

```
class SportController extends Controller  
{  
    public function sports(Request $request)  
    {  
        return Sport::all(); } }
```

14.2.5. StudentController

Този контролер, **StudentController**, се грижи за обработка на различни заявки, свързани със студентите. Ето кратко обяснение на всеки от методите в него:

1. **getGrant(Request \$request)**: Връща заявката за стипендия на текущия потребител (студент), ако такава съществува.
2. **setGrant(Request \$request)**: Създава нова заявка за стипендия за текущия потребител и изпраща известие за успешното кандидатстване.
3. **getDorms(Request \$request)**: Връща всички свободни общежития, в които студентът може да се засели.
4. **getDorm(Request \$request)**: Връща общежитието, в което е заселил текущия потребител.
5. **setDorm(Request \$request)**: Задава общежитието на текущия потребител.
6. **getSport(Request \$request)**: Връща спортната дейност, която е записана за текущия потребител.

7. **setSport(Request \$request):** Създава запис за спортна дейност за текущия потребител и изпраща известие за успешното записване.
8. **grades(Request \$request):** Връща оценките на текущия потребител, групирани по семестри.

```
class StudentController extends Controller

{
    public function getGrant(Request $request)
    {
        return GrantRequest::where("student_id",
        $request->user()->id)->first();
    }

    public function setGrant(Request $request)
    {

        NotificationService::sendPushNotification($request->user()->push_token,
        "Стипендия", "Успешно кандидатстване");

        return GrantRequest::create([
            "point" => $request->point,
            "student_id" => $request->user()->id
        ]);
    }

    public function getDorms(Request $request)
    {
        return Dorm::whereNull("student_id")->get();
    }
}
```

```

public function getDorm(Request $request)
{
    NotificationService::sendPushNotification($request->user()->push_token,
    "Общежития", "Успешно кандидатстване");

    return Dorm::where("student_id", $request->user()->id)->first();
}

public function setDorm(Request $request)
{
    $dorm = Dorm::where("id", $request->dorm_id)->first();

    $dorm->student_id = $request->user()->id;

    return $dorm->save();
}

public function getSport(Request $request)
{
    $sport = $request->user()->sport->first();

    if ($sport != null) {
        $sport->teacher;
    }

    return $sport;
}

public function setSport(Request $request)
{
    NotificationService::sendPushNotification($request->user()->push_token,
    "Спорт", "Успешно записване");
}

```

```

        return StudentSport::create([
            "student_id" => $request->user()->id,
            "sport_id" => $request->sport["id"]
        ]);
    }

    public function grades(Request $request)
    {
        $semesters = [];
        for ($i = 1; $i < $request->user()->semester + 1; $i++) {
            $grades = Grade::where("student_id",
                $request->user()->id)->get();
            $grades_f = [];
            foreach ($grades as $grade) {
                $grade->teacher;
                if ($grade->discipline->semester == $i) {
                    $grades_f[] = $grade;
                }
            }
            $semesters[] = $grades_f;
        }
        return $semesters;
    }
}

```

14.2.6. TeacherController

Този контролер, **TeacherController**, се грижи за обработка на заявки, свързани с учителите. Ето обяснение на методите в него:

1. **sport(Request \$request)**: Създава нов спорт и го свързва с текущия потребител (учител). Стойността на полето "quantity" е фиксирана на 10.
2. **students(Request \$request)**: Връща всички потребители с роля "студент".
3. **grade(Request \$request)**: Създава нова оценка за ученика, свързана с текущия потребител (учител) и дисциплината. След създаването на оценката, изпраща известие до ученика за новата оценка.

```
class TeacherController extends Controller
{
    public function sport(Request $request)
    {
        return Sport::create([
            "name" => $request->name,
            "teacher_id" => $request->user()->id,
            "quantity" => 10
        ]);
    }

    public function students(Request $request)
    {
        return User::where("role", "student")->get();
    }

    public function grade(Request $request)
    {
        $grade = Grade::create([
            "grade" => $request->grade,
            "verified" => $request->verified,
        ]);
    }
}
```

```

        "student_id" => $request->student_id,
        "teacher_id" => $request->user()->id,
        "discipline_id" => $request->discipline_id
    ] );
}

$discipline = Discipline::where("id",
$grade->discipline_id)->first();

$title = "Нова Оценка по " . $discipline->name;
$body = "Оценка: " . $grade->grade;

NotificationService::sendPushNotification($grade->student->push_token,
$title, $body); }

```

14.2.7. UserController

Този контролер, **UserController**, се грижи за обработка на заявки, свързани с потребителите. Всичко:

1. **login(Request \$request)**: Проверява дали потребителят със зададения факултетен номер и ЕГН съществува в системата. Ако потребителят съществува, актуализира неговият push токен (ако е наличен) и го връща. Ако не е намерен потребител, връща false.
2. **authlogin(Request \$request, \$token)**: Извършва автентикация на потребителя, използвайки API токена му.
3. **profile(Request \$request)**: Връща профила на текущия потребител. Ако той е студент, допълнително зарежда групата, специалността и факултета му.

```

class UserController extends Controller
{
    public function login(Request $request)
    {
        Log::debug("Facnum: " . $request->facnum);
    }
}

```

```

Log::debug("Egn: " . $request->egn);

$user = User::where("facnum", $request->facnum)->where("egn",
$request->egn)->first();

if ($user != null) {
    Log::debug("User Found");
} else {
    Log::debug("! User NOT Found !");
    return false;
}

Log::debug("LOGGED IN");
Log::debug("LOGGED IN WITH PUSH TOKEN: " . $request->pushToken);

if (strlen($request->pushToken) != 0) {
    $user->push_token = $request->pushToken;
    $user->save();
}

return $user;
}

public function authlogin(Request $request, $token)
{
    return User::where("api_token", $token)->first();
}

```

```

    }

    public function profile(Request $request)
    {
        $user = $request->user();

        if ($user->role == "student") {
            $user->group = Group::where("id", $user->group_id)->first();
            $user->group->specialty = Specialty::where("id",
$user->group->specialty_id)->first();
            $user->group->specialty->faculty = Faculty::where("id",
$user->group->specialty->faculty_id)->first();
        }
        return $user;
    }
}

```

14.3. Сървиси

14.3.1.

Този клас, **NotificationService**, предоставя метод **sendPushNotification**, който се използва за изпращане на известия чрез **push** уведомления. Той изпраща **POST** заявка към **API** на **Expo** (платформа за управление на **push** уведомления), като предоставя получателя (**\$to**), заглавието (**\$title**) и съдържанието (**\$body**) на уведомлението.

```

class NotificationService
{
    static function sendPushNotification($to, $title, $body)
    {
        $client = new Client();
        $client->request("POST", "https://exp.host/--/api/v2/push/send", [

```

```

'headers' => [
    "Accept" => "application/json",
    "Accept-encoding" => "gzip, deflate",
    "Content-Type" => "application/x-www-form-urlencoded",
] ,
'form_params' => [
    "to" => $to,
    "title" => $title,
    "body" => $body,
]
] );
} }

```

14.4. Мидълуер

14.4.1. Authenticate

Този клас, **Authenticate**, е **middleware** (посредник), който се използва за аутентикация на потребители. Методът **redirectTo** се извиква, когато потребителят не е аутентициран и се опитва да достъпи ресурс, който изисква аутентикация. В случая, кодът просто извиква функцията **abort(401)**, която генерира HTTP отговор с код за грешка 401 (Неавторизиран), указвайки на клиента, че не е оторизиран за достъп до съответния ресурс.

```

class Authenticate extends Middleware
{
    protected function redirectTo($request)
    {
        abort(401);
    }
}

```

14.4.2. RedirectIfAuthenticated

Този клас, `RedirectIfAuthenticated`, представлява **middleware**, който се използва за пренасочване на аутентицираните потребители. Методът `handle` се изпълнява при всяка входяща заявка и проверява дали потребителят е аутентициран за един или повече "guard" контекста. Ако потребителят е аутентициран за някой от зададените "guard" контексти, се пренасочва към домашната страница на приложението. В противен случай, заявката се препраща към следващия **middleware** в веригата.

```
class RedirectIfAuthenticated

{

    /**
     * Handle an incoming request.
     *
     * @param \Illuminate\Http\Request $request
     * @param \Closure $next
     * @param string[]|null ...$guards
     * @return mixed
     */

    public function handle($request, Closure $next, ...$guards)
    {

        $guards = empty($guards) ? [null] : $guards;

        foreach ($guards as $guard) {

            if ($Auth::guard($guard)->check()) {

                return redirect(RouteServiceProvider::HOME);

            }
        }

        return $next($request); }

}
```

15. Фронтенд

15.1. Assets

Директорията "assets" съдържа основните ресурси, използвани в приложението, като снимки, векторни изображения, шаблонни фигури и други подобни елементи.

adaptive-icon.png - изображението на началната страница на приложението

favicon.png - иконата, която се зарежда при получаване на нотификация,

15.2. Компоненти

15.2.1. Form

Този код представлява функционален React компонент, наречен "**Input**", който създава текстово поле в **React Native** приложение. Компонентът приема пропъртита като параметри и се състои от едно текстово поле, вложено в контейнер. Стиловете са дефинирани чрез функцията "**StyleSheet.create()**" и определят външния вид на текстовото поле и контейнера около него.

```
export default function Input(props) {
  return (
    <View style={styles.view}>
      <TextInput style={styles.input}
        onChangeText={props.onChangeText}
        secureTextEntry={props.secureTextEntry}
        defaultValue={props.defaultValue}
        placeholder={props.title}/>
    </View>
  );
}

const styles = StyleSheet.create({
  view: {
    width: '100%',
    height: 40,
    padding: 5,
    margin: 10,
    border: 1px solid #ccc,
    borderRadius: 5
  },
  input: {
    width: '100%',
    height: 100%
  }
});
```

```
) ;  
}  
  
const styles = StyleSheet.create({  
  view: {  
    display: "flex",  
    flexDirection: "column",  
    alignContent: "center",  
    justifyContent: "center",  
    alignItems: "center",  
    width: 200,  
  },  
  text: { },  
  input: {  
    borderBottomColor: "#000",  
    borderLeftColor: "#fff",  
    borderTopColor: "#fff",  
    borderRightColor: "#fff",  
    borderWidth: 2,  
    height: 40,  
    width: 200,  
    backgroundColor: "#fff",  
  },  
});
```

15.2.2. Auth Navigator

Дефинираме функция, която създава различни навигационни екрани в зависимост от състоянието на аутентикацията на потребителя. Използва се библиотеката "Drawer Navigator" за създаване на навигационен чекмедже. За всеки екран се задават определени настройки за заглавие и стил на хедъра. Ако потребителят не е аутентициран, се предлага екран за вход, а в противен случай се показват различни екрани в зависимост от ролята на потребителя - "student" или "teacher".

```
const Drawer = createDrawerNavigator();

const navStyles = {
  headerStyle: {
    backgroundColor: "#3352A8",
  },
  headerTintColor: "#fff",
  headerTitleStyle: {
    fontWeight: "bold",
  },
};

export function getAuthNavigator(auth, pushToken) {
  if (!auth) {
    return (
      <>
        <Drawer.Screen
          name="Login"
          component={LoginScreen}
          options={{
            headerStyle: {
              backgroundColor: "#3352A8",
            },
            headerTintColor: "#fff",
            headerTitleStyle: {
              fontWeight: "bold",
            },
          }}
        />
    );
  }
}
```

```
        title: "Вход",
        ...navStyles,
    } }

initialParams={{ pushToken }}
```

```
</>

) ;

}

if (auth) {

    if (auth.role == "student") {

        return (
            <>
                <Drawer.Screen
                    name="Profile"
                    component={ProfileScreen}
                    options={{

                        title: "Профиль",
                        ...navStyles,
                    }}>
            />
                <Drawer.Screen
                    name="Grades"
                    component={StudentGradesScreen}
                    options={{

                        title: "Оценки",

```

```
    ...navStyles,  
  } }  
  />  
  
<Drawer.Screen  
  name="Calendar"  
  component={StudentCalendarScreen}  
  options={ {  
    title: "Календар",  
    ...navStyles,  
  } }  
  />  
  
<Drawer.Screen  
  name="Logout"  
  component={LogoutScreen}  
  options={ {  
    title: "Изход",  
    ...navStyles,  
  } }  
  />  
  
</>  
);  
}  
  
if (auth.role == "teacher") {  
  return (  
    <>
```

```
<Drawer.Screen  
    name="Profile"  
    component={ProfileScreen}  
    options={ {  
        title: "Профиль",  
        ...navStyles,  
    } }  
/>  
  
<Drawer.Screen  
    name="Grades"  
    component={TeacherGradesScreen}  
    options={ {  
        title: "Оценки",  
        ...navStyles,  
    } }  
/>  
  
<Drawer.Screen  
    name="Logout"  
    component={LogoutScreen}  
    options={ {  
        title: "Изход",  
        ...navStyles,  
    } }  
/>  
</>
```

```
) ;  
}  
}  
}
```

15.3. Контекст

15.3.1. AuthContext

Използваме функцията `createContext` от библиотеката `react` за създаване на контекст за управление на аутентикацията в React приложение.

Променливата `defaultAuthValue` представлява стойност по подразбиране за контекста за аутентикация. Тя е масив с два елемента: първият елемент е `null`, който обозначава, че няма аутентикация, а вторият елемент е празна функция, която не прави нищо.

След това се създава самият контекст за аутентикация чрез `createContext(defaultAuthValue)`. Този контекст ще се използва за предаване на информация за състоянието на аутентикацията на различните компоненти в приложението.

```
import { createContext } from "react";  
  
export const defaultAuthValue = [null, () => {}];  
  
export const AuthContext = createContext(defaultAuthValue);
```

15.3.2. Push Context

Представяме стойности за управление на Push нотификации. Контекстът е инициализиран със стойност по подразбиране - масив, съдържащ `null` и празна функция. Компонентите могат да използват този контекст, за да получават или

променят стойностите на Push уведомленията без да се налага да пренасят данните чрез прокси.

```
import { createContext } from "react";

export const defaultAuthValue = [null, () => {}];

export const PushContext = createContext(defaultAuthValue);
```

15.4. Екрани

15.4.1. StudentCalendarScreen

Правим функционален компонент, наречен **StudentCalendarScreen**, който използва **useState** за управление на състоянието. Компонентът включва два календара - единият показва графика за три дни, а другият показва периодични събития. Височините на календарите се актуализират чрез **useState**. Функцията **handleLayout** се изпълнява при събитие **onLayout** и актуализира височините на календарите. В компонента се използва масив от събития, който съхранява заглавия и дати.

```
export default function StudentCalendarScreen() {

    const [calendarHeight, setCalendarHeight] = useState(0);

    const [scheduleHeight, setScheduleHeight] = useState(0);

    const [layoutChanged, setLayoutChanged] = useState(false);

    function handleLayout(e) {

        if (layoutChanged) {

            return

        }

        const { width, height } = e.nativeEvent.layout;

        console.log(height);

        setCalendarHeight(70/100*height)

    }

}
```

```

        setScheduleHeight(20/100*height)

        setLayoutChanged(true)

    }

const events = [
] .

return (
    <View onLayout={handleLayout}>

        <Calendar events={events} height={calendarHeight} mode="3days"
swipeEnabled={false} scrollOffsetMinutes={420} />

        <Calendar events={events} height={scheduleHeight}
mode="schedule" />    </View>

```

15.4.2. StudentDormScreen

Този код представлява **React Native** компонент, който позволява на студентите да кандидатстват за общежитие. Компонентът извиква **API** заявки, за да получи информация за наличните общежития и за вече кандидатстваното общежитие на студента. В зависимост от това дали студентът вече е кандидатствал, се показва съобщение или форма за кандидатстване. Формата включва списък с наличните общежития и бутон за кандидатстване.

```

export default function StudentDormsScreen() {

    const [selectedDorm, setSelectedDorm] = useState();

    const [dorms, setDorms] = useState();

    const [dorm, setDorm] = useState();




    const api = useApi();




    async function getDorm() {

```

```
const res = (await api.get("/student/dorm")).data;
setDorm(res);
}

async function getDorms() {
  const res = (await api.get("/student/dorms")).data;
  setSelectedDorm(res[0])
  setDorms(res);
}

async function handlePress() {
  api.post("/student/dorm", {
    dorm_id: selectedDorm,
  });
}

useFocusEffect(
  useCallback(() => {
    getDorm();
    getDorms();
  }, [])
);

return (
  <View>
```

```

<View style={{ padding: 20 }}>

  {dorm ? (
    <>
      <Text>Вече сте кандидатствали за общежитие:
{dorm.number}</Text>
    </>
  ) : (
    <>
      <Text>Кандидатстване за общежитие:
{selectedDorm?.number}</Text>
      <Picker selectedValue={selectedDorm}
onValueChange={(itemValue, itemIndex) => setSelectedDorm(itemValue)}>
        {dorms?.map((mdorm, index) => (
          <Picker.Item key={`dorm-${index}`}
label={`Блок: ${mdorm.block} Стая: ${mdorm.number}`} value={mdorm} />
        )));
      </Picker>
      <Button color={primaryColor} title="Кандидатствай"
onPress={handlePress}></Button>
    </>
  ) }
</View>
</View>)

```

15.4.3. StudentEventScreen

Този код представлява **React Native** компонент **StudentEventsScreen**, който използва календарен компонент за показване на събития за студенти. Чрез извикване на API заявка в **getEvents()**, се зареждат реални събития от сървъра, които след това се обработват и се задават като състояние **events**. Функцията **useFocusEffect** се използва, за да се гарантира, че при всяко фокусиране на екрана се актуализира списъкът със събития. Компонентът връща **JSX** с **Calendar** компонентът, който показва събитията.

```
export default function StudentEventsScreen() {  
  const [events, setEvents] = useState([  
    {  
      title: "",  
      start: new Date(2024, 3, 23, 7, 30),  
      end: new Date(2024, 3, 23, 7, 30),  
    },  
  ]);  
  
  const api = useApi();  
  
  async function getEvents() {  
    const res = (await api.get("/events")).data;  
  
    for (const ev of res) {  
      ev.start = new Date(ev.start);  
    }  
  }  
}
```

```

    ev.end = new Date(ev.end);

}

setEvents(res);

}

useFocusEffect(
    useCallback(() => {
        getEvents();
    }, [])
);

return (
    <View>
        <Calendar events={events} height={800} mode="schedule" />
    </View>
);
}

```

15.4.4. StudentGradeScreen

StudentGradeScreen, показва оценките на студента и средния му успех. Използва се **useState** за управление на състоянието, **useApi** за вземане на данни от сървъра, и **useFocusEffect** и **useCallback** за зареждане на оценките само при първоначалното зареждане на екрана. Визуализира се списък с оценки за всеки семестър и средния

успех на студента. Стиловете се използват за подчертаване на визуалния аспект на данните.

```
export default function StudentGradesScreen() {  
  
  const [semesters, setSemesters] = useState([]);  
  
  const [averageGrade, setAverageGrade] = useState(0);  
  
  
  const api = useApi();  
  
  async function getGrades() {  
  
    const data = (await api.get("student/grades")).data;  
  
  
    let gradeSum = 0;  
    let gradeNum = 0;  
  
  
    for (const semester of data) {  
  
      for (const grade of semester) {  
  
        gradeSum += grade.grade;  
  
        gradeNum++;  
  
      }  
  
    }  
  
    setAverageGrade((gradeSum / gradeNum).toFixed(2));  
  
    setSemesters(data);  
  
  }  
  
  useFocusEffect(useCallback(() => {  
  
    getGrades();  
  
  }, []));  
  
  return (  
}
```

```

<ScrollView>

    <View style={{ display: "flex", justifyContent: "center",
alignItems: "center", height: 100 }}>

        <Text style={{ backgroundColor: "#50C878", borderStyle:
"solid", borderRadius: 100, paddingHorizontal: 10, paddingVertical: 2,
color: "#fff" }}>Среден успех (за всички години): {averageGrade}</Text>

    </View>

    {semesters?.map((grades, semesterIndex) => (

        <View style={{ backgroundColor: semesterIndex % 2 == 0 ?
"#fff" : "#eee" }} key={`semester-${semesterIndex}`}>

            <Text style={styles.semesterText}>Семестър:
{semesterIndex + 1}</Text>

            <View style={styles.grades}>

                {grades?.map((grade, gradeIndex) => (
                    <View style={styles.info}
key={`grade-${gradeIndex}`}>

                        <View style={styles.infoBall}></View>

                        <View>

                            <Text>Дисциплина:
{grade.discipline.name}</Text>

                            <Text>Оценка: {grade.grade}</Text>

                            <Text>Нанесена от:
{grade.teacher.name}</Text>

                            <Text style={{ color: grade.verified ? "green" : "red" }}>{grade.verified ? "Заверен" : "Незаверен"}</Text>
                        </View>
                </View>
            </View>
        </View>
    </View>
)

```

```
        ) ) }

      </View>

    </View>

  ) ) }  </ScrollView> ) ; }
```

15.4.5. StudentGrantScreen

Този **React Native** компонент позволява на студентите да кандидатстват за стипендия. В него се използват три състояния: **point**, **averageGrade**, и **grant**. Функцията **getGrant()** зарежда вече кандидатстваната стипендия от сървъра, а **getGrades()** изчислява средния успех на студента. При натискане на бутона за кандидатстване, функцията **handlePress()** прави **POST** заявка към API с избраната точка за стипендия и актуализира състоянието **grant**. Използва се **useFocusEffect**, за да се извикат функциите **getGrant()** и **getGrades()** при фокусиране на екрана. В зависимост от състоянието на **grant**, се показва съобщение за вече кандидатствана стипендия или форма за кандидатстване, включваща средния успех и бутон за кандидатстване.

```
export default function StudentGrantScreen() {

  const [point, setPoint] = useState(3);

  const [averageGrade, setAverageGrade] = useState(0);

  const [grant, setGrant] = useState();

  const api = useApi();

  async function getGrant() {

    const res = (await api.get("/student/grant")).data;

    setGrant(res);

  }

}
```

```
async function getGrades() {  
  
    const data = (await api.get("student/grades")).data;  
  
  
    let gradeSum = 0;  
    let gradeNum = 0;  
  
  
    for (const semester of data) {  
  
        for (const grade of semester) {  
  
            gradeSum += grade.grade;  
  
            gradeNum++;  
  
        }  
    }  
  
  
    setAverageGrade((gradeSum / gradeNum).toFixed(2));  
}  
  
  
async function handlePress() {  
  
    api.post("/student/grant", {  
  
        point,  
  
    }) ;  
  
    setGrant({ point });  
  
}  
  
  
useFocusEffect(  
  
    useCallback(() => {
```

```

        getGrant();

        getGrades();

    } , [ ])

);

return (
<View>

<View style={{ padding: 20 }}>

{grant ? (
<>

<Text>Вече сте кандидатствали по точка:
{grant.point}</Text>

</>

) : (
<>

<Text>Кандидатстване за стипендия: {point}</Text>

<View style={{ display: "flex", justifyContent:
"center", alignItems: "center", height: 100 }}>

<Text style={{ backgroundColor: "#50C878",
borderStyle: "solid", borderRadius: 100, paddingHorizontal: 10,
paddingVertical: 2, color: "#fff" }}>Среден успех (за всички години):
{averageGrade}</Text>

</View>

<Picker selectedValue={point}
onValueChange={(itemValue, itemIndex) => setPoint(itemValue)}>

<Picker.Item label="Точка 3" value="3" />

</Picker>

<Button color={primaryColor} title="Кандидатствай"
onPress={handlePress}></Button>

```

```
        </>

    ) }

</View>

</View>) ; }
```

15.4.6. TeacherInsuranceScreen

StudentInsuranceScreen е React Native компонент, който показва информация за застраховки, свързани с ученическо/студентско осигуряване. Чрез `useApi` се извиква заявка за зареждане на застраховките. Получената информация се показва в таблица с помощта на **DataTable** компонента.

```
export default function StudentInsuranceScreen() {

    const [insurances, setInsurances] = useState([]);

    const api = useApi();

    async function getInsurances() {

        const res = (await api.get("/insurances")).data;

        setInsurances(res);
    }

    useEffect(
        useCallback(() => {
            getInsurances();
        }, [])
    );
}

return (
    <ScrollView>
```

```

<DataTable>

    <DataTable.Header>
        <DataTable.Title>Година</DataTable.Title>
        <DataTable.Title>Месец</DataTable.Title>
        <DataTable.Title numeric>Осиг. сума</DataTable.Title>
        <DataTable.Title numeric>Осигуровка</DataTable.Title>
    </DataTable.Header>

    {insurances?.map((insurance, index) => (
        <DataTable.Row key={`${insurance}-${index}`}>
            <DataTable.Cell>{insurance.year}</DataTable.Cell>
            <DataTable.Cell>{insurance.month}</DataTable.Cell>
            <DataTable.Cell
                numeric>{insurance.insurance_sum}</DataTable.Cell>
            <DataTable.Cell
                numeric>{insurance.insurance}</DataTable.Cell>
        </DataTable.Row>
    )));
}

</DataTable>
</ScrollView>);
}

```

15.4.7. TeacherSportScreen

StudentSportScreen е компонент, който позволява на студентите да се записват за спорт. Извикват API заявки за получаване на списък със спортове и информация за вече записания спорт на студента. Студентът може да избере спорт от списъка, след което се изпраща заявка към чрез `handlePress()`, за да се запише за избрания спорт. Функцията `useFocusEffect` гарантира, че заявките към API се извикват при фокусиране на екрана. В зависимост от това дали студентът вече е записан за

спорт, се показва съответно информация за вече записания спорт или форма за избор на нов спорт.

```
export default function StudentSportScreen() {  
  
  const [selectedSport, setSelectedSport] = useState();  
  
  const [sports, setSports] = useState();  
  
  const [sport, setSport] = useState();  
  
  
  const api = useApi();  
  
  
  async function handlePress() {  
  
    await api.post("/student/sport", { sport: selectedSport });  
  
  }  
  
  async function getSport() {  
  
    const res = (await api.get("/student/sport")).data;  
  
    setSport(res);  
  
  }  
  
  
  async function getSports() {  
  
    const res = (await api.get("/sports")).data;  
  
    setSports(res);  
  
    setSelectedSport(res[0]);  
  
  }  
  
  
  useFocusEffect(  
  
    useCallback(() => {
```

```

        getSports();

        getSport();

    }, []
);

return (
<View>

{sport ? (
<View style={{ padding: 20, gap: 8 }}>
    <Text>Вече сте записан на спорт: {sport.name} с
ръководител: {sport.teacher.name}</Text>
</View>
) : (
<View style={{ padding: 20, gap: 8 }}>
    <Text>Избери Спорт</Text>
    <View>
        <Picker selectedValue={selectedSport}
onValueChange={(itemValue, itemIndex) => setSelectedSport(itemValue)}>
            {sports?.map((msport, index) => (
<Picker.Item key={`sport-${index}`}
label={`${msport.name}`} value={msport} />
            )));
        </Picker>
    </View>
    <Button color={primaryColor} title="Избери"
onPress={handlePress}></Button>
</View>
)
);

```

```
        ) }  
  
    </View> ) ; }
```

15.4.8. TeacherGradeDisciplineScreen

Предоставя функционалност за влизане в системата. Използва се контекстен хук, за да се достъпят стойностите на **AuthContext** и **PushContext**. Потребителят може да влезе в системата чрез факултетен номер и ЕГН, както и да удостовери входа си чрез биометрични данни. При наличие на запазен API токен, входът става автоматично. Използват се компоненти като **View**, **Text**, **Input**, **Image** и **Button** за създаване на UI. Екранът се стилизиран

```
export default function TeacherGradesDisciplineScreen(props) {  
  
    const api = useApi();  
  
    const [auth, setAuth] = useContext(AuthContext);  
  
    const [students, setStudents] = useState([]);  
  
  
    const navigation = useNavigation();  
  
  
    async function getStudents() {  
  
        const data = (await api.get("/teacher/students")).data;  
  
        setStudents(data);  
  
    }  
  
  
    async function handleAdd(student) {  
  
        await api.post("/teacher/grade", {  
  
            grade: student.grade,  
  
            verified: true,
```

```

        student_id: student.id,
        discipline_id: student.discipline.id,
    }) ;
}

async function handleChange(text, student) {
    student.grade = text;
    student.discipline = props.route.params.discipline;
}

useEffect(() => {
    getStudents();
}, []);

return (
<View style={{ padding: 10, display: "flex", gap: 10 }}>
    {students?.map((student, index) => (
        <View key={`student-${index}`} style={{ display: "flex",
flexDirection: "row", justifyContent: "center", alignItems: "center", gap: 4 }}>
            <Text>{student.name}</Text>
            <Text>Оценка:</Text>
            <TextInput style={{ width: 70, height: 50,
backgroundColor: "#fff" }} keyboardType="numeric" onChangeText={(text) =>
handleChange(text, student)} />
            <Button key={`student-${index}`} title={"Добави"}
onPress={() => handleAdd(student)}></Button>
    
```

```
</View>      )) } </View> ) ; }
```

15.4.9. TeacherGradeHomeScreen

TeacherGradesHomeScreen, показва списък с дисциплини, които учителят преподава. Списъкът се извлича от API чрез хукът `useApi` и се съхранява в състоянието `disciplines`. При монтаж на компонента се извиква функцията `getDisciplines`, която извлича дисциплините. При натискане на бутон за дисциплина се извиква функцията `routeToDiscipline`, която навигира потребителя към екрана на съответната дисциплина.

```
export default function TeacherGradesHomeScreen() {  
  
  const api = useApi();  
  
  const [disciplines, setDisciplines] = useState([]);  
  
  
  const navigation = useNavigation();  
  
  
  async function getDisciplines() {  
  
    const data = (await api.get("/disciplines")).data;  
  
    setDisciplines(data);  
  }  
  
  
  function routeToDiscipline(discipline) {  
  
    navigation.navigate("Discipline", { discipline });  
  }  
  
  
  useEffect(() => {  
  
    getDisciplines();  
  }, []);
```

```
return (

  <View style={{ padding: 10, display: "flex", gap: 10 }}>

    {disciplines?.map((discipline, index) => (
      <Button key={`discipline-${index}`} title={discipline.name}
onPress={() => routeToDiscipline(discipline)}></Button>
    )));
  </View> );
}
```

15.4.10. TeacherGradeScreen

Този код създава навигационен стек в **React Native** с два екрана: "**Home**" и "**Discipline**". Всеки екран съответства на различен компонент. "**Home**" екранът показва списък с дисциплини, докато "**Discipline**" показва подробности за конкретна дисциплина.

```
const Stack = createNativeStackNavigator();

export default function TeacherGradesScreen() {
  return (
    <Stack.Navigator>

      <Stack.Screen name="Home" component={TeacherGradesHomeScreen}
options={{ title: "Дисциплини" }} />

      <Stack.Screen name="Discipline"
component={TeacherGradesDisciplineScreen} options={{ title: "Дисциплина" }} />

    </Stack.Navigator>
  );
}
```

15.4.11. LoginScreen

Този код представлява еcran за вход в приложението в React Native. В него се използват основни React хуци като `useState`, `useContext` и `useEffect`, за управление на състоянието и ефектите. Компонентът показва изображение на логото на ТУ София и полета за въвеждане на потребителско име и парола. При вход или удостоверяване на потребителя се извикват съответните функции. Грешките се показват в червен текст под формата на текстово съобщение.

```
export default function LoginScreen(props) {

    const [, setAuth] = useContext(AuthContext);

    const [pushToken] = useContext(PushContext);

    const [facnum, setFacnum] = useState("");
    const [egegn, setEgegn] = useState("");
    const [savedToken, setSavedToken] = useState(false);
    const [error, setError] = useState(null);

    const api = useApi();

    async function authLogin() {

        try {

            const auth = await LocalAuthentication.authenticateAsync();

            if (auth.success) {

                let user = (await
api.get(`authlogin/${savedToken}`)).data;

                console.log(user);

                if (!user) throw new Error();

                setAuth(user);
            }
        } catch (err) {
            setError(err.message);
        }
    }
}
```

```
        }

    } catch (error) {

        await AsyncStorage.clear()

        setSavedToken(false)

        console.error(error);

        return;

    }

}

async function login() {

    let user = {};

    try {

        user = (await api.post("/login", { facnum, egn, pushToken
})).data;

    } catch (error) {

        setError("Няма Връзка с Бекенд");

        console.error(error);

        return;

    }

    if (!user) {

        setError("Грешно Име или Парола");

    } else {

        if (user.api_token.length == 80) {

            await AsyncStorage.setItem("token", user.api_token);

            setAuth(user);

        }

    }

}
```

```
}

async function checkSavedToken() {
    setSavedToken(await AsyncStorage.getItem("token"));
}

useEffect(() => { }, [pushToken]);
useEffect(() => {
    checkSavedToken();
}, []);

return (
    <View style={styles.container}>
        {error ? <Text style={{ color: "red" }}>{error}</Text> : <></>}
        <Image source={tuLogo} style={styles.img}></Image>
        <Input title={"Име"} onChangeText={(t) => setFacnum(t)}
defaultValue={facnum} />
        <Input title={"Парола"} onChangeText={(t) => setEgn(t)}
defaultValue={egn} secureTextEntry />
        <Button title="Вход" color="#3352A8" onPress={login} />
        {savedToken && <Button title="Удостоверен Вход" color="#444"
onPress={authLogin} />}
    </View>);
}
```

15.4.12. LogoutScreen

Този код представлява еcran за изход от приложението . При монтаж на компонента, той изчиства всички данни от **AsyncStorage**, включително потребителския токен за автентикация, и нулира състоянието на автентикацията до **null**. Няма визуализация на екрана, само логика за изход.

```
export default function LogoutScreen(props) {  
  
  const [auth, setAuth] = useContext(AuthContext);  
  
  useEffect(() => {  
  
    AsyncStorage.clear();  
  
    setAuth(null);  
  
  }, []);  
  
  return <View></View>;  
  
}
```

15.4.13. TeacherGradeScreen

Този код представлява еcran за профил на потребител. Използват се хуковете за управление на състоянието и ефектите. При зареждане на компонента се изпраща заявка към API за вземане на данни за профила на студента. Визуализират се името на студента и логото на университета, както и различни характеристики на профила.

```
const tuLogo = require("../assets/TUSlogosimple.png");  
  
export default function ProfileScreen() {
```

```

const [auth, setAuth] = useContext(AuthContext);

const [student, setStudent] = useState(null);

const api = useApi();

async function getData() {
  try {
    const data = await (await api.get("/profile")).data;
    setStudent(data);
  } catch (error) {
    console.error(error);
  }
}

useEffect(() => {
  getData();
}, []);

function existRender(text, key) {
  if (key) {
    return (
      <View style={styles.info}>
        <View style={styles.infoBall}></View>
        <Text style={styles.infoText}>
          {text}: {key}
        </Text>
      </View>
    );
  }
}

```

```

    ) ;

} else return <></>;

}

return (
<ScrollView>

<View style={styles.topContainer}>
<Text style={styles.text}>{student?.name}</Text>
<Image style={styles.topImage} source={tuLogo} />
</View>
<View style={styles.infoContainer}>
{existRender("OKC", student?.eqd) }

{existRender("Факултетен номер", student?.facnum) }

{existRender("Факултет",
student?.group?.specialty?.faculty?.name) }

{existRender("Специалност",
student?.group?.specialty?.name) }

{existRender("Вид обучение", student?.eqd_type) }

{existRender("Група", student?.group?.id) }

{existRender("Състояние", student?.state) }

{existRender("Записан семестър", student?.semester) }

{existRender("E-mail", student?.mail) }
</View>
</ScrollView>
) ; } ;

```

16. Потребителски интерфейс и ръководство

16.1. Цветове и дизайн

Потребителският интерфейс използва бял фон и сапфирен син цвят за основен, който може да се свърже с Техническия университет София.

Навигацията на приложението е организирана посредством чекмеджета, което предоставя удобен и интуитивен начин за потребителите да откриват различни функции и секции на приложението.

Изгледите на приложението са текстови и таблични.,

16.2. Вход в приложението

При стартиране на приложението потребителите се посрещат от форма за влизане. След като изберат тази форма, полетата за въвеждане на данни автоматично се подреждат така, че да бъде по-лесно за тях да прегледат и въведат необходимата информация. Ако потребителят вече е влязъл в своя профил, се предлага удобна опция за вход, като използване на пръстов отпечатък или разпознаване на лиц.

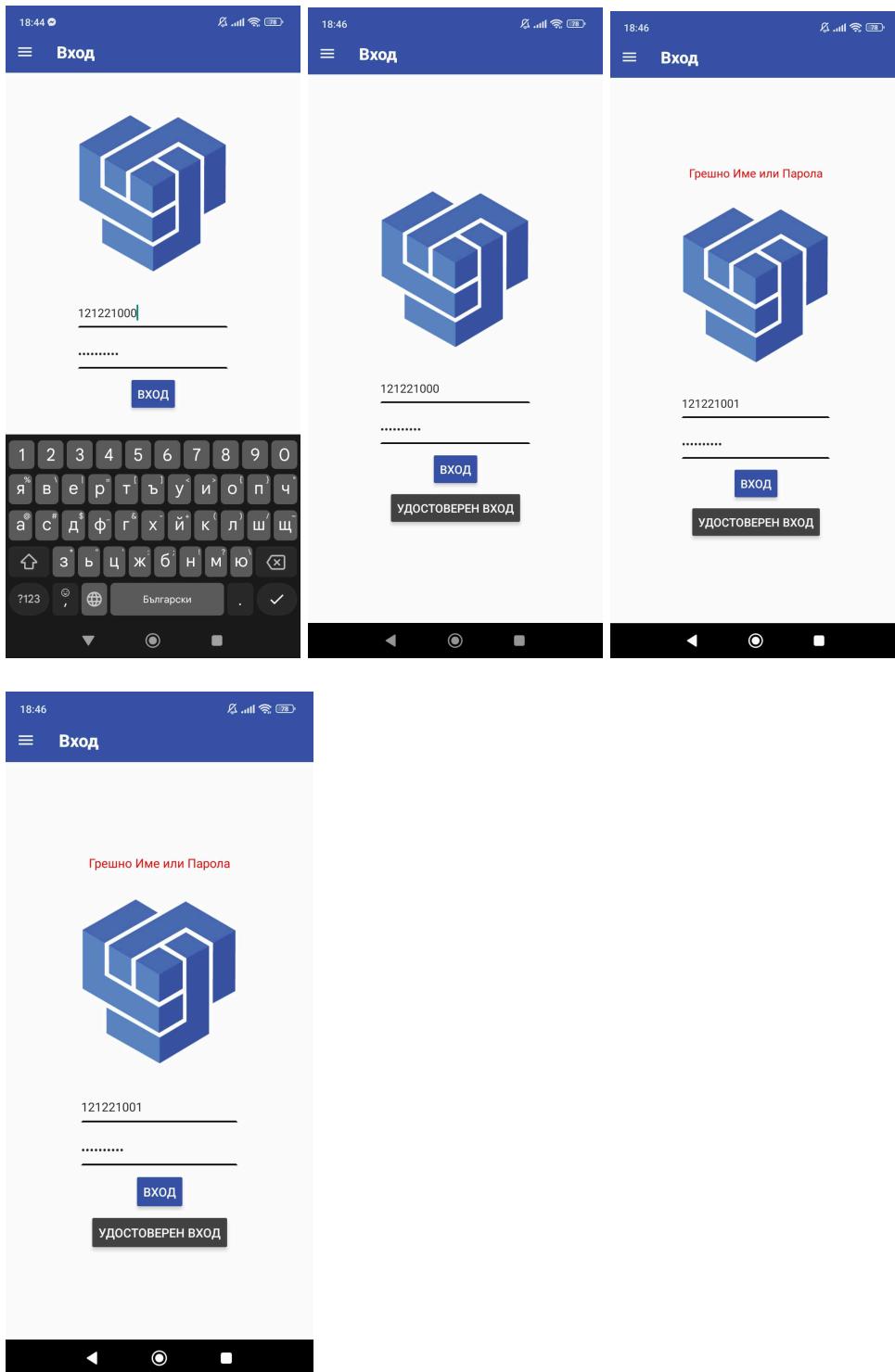
В случай на грешно въведени данни потребителите получават съобщение за грешка.

Интерфейсът е показан на снимки от 16.1.1 до 16.1.4

Фиг 16.1.1

Фиг 16.1.2

фиг 16.1.3



Фиг 16.1.4

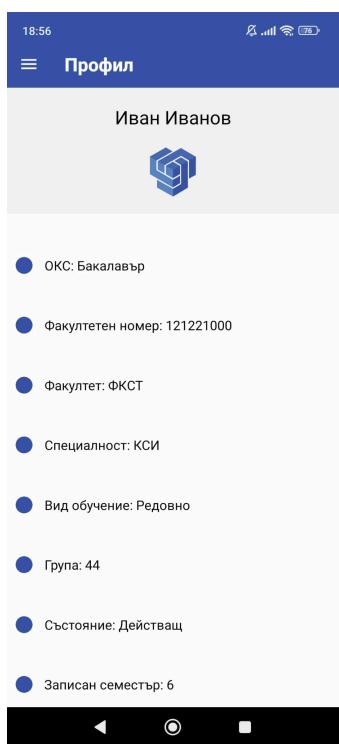
16.3. Информация за студена

След успешно влизане като студент, потребителя е посрещнат с информация за себе си.

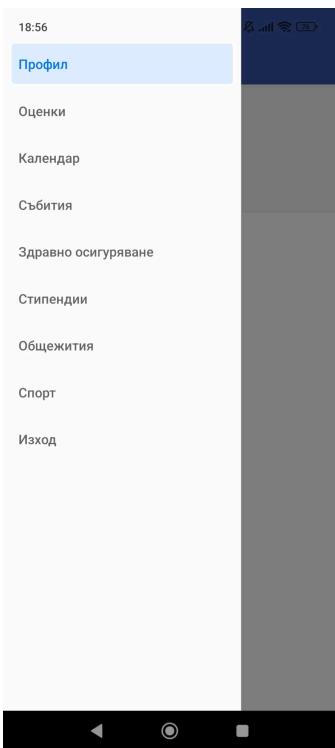
От тази страница той лесно може да се придвижи до останалите чрез навигацията тип чекмедже.

Интерфейсът е показан на фигури 16.3.1 и 16.3.2

Фиг. 16.3.1



Фиг. 16.3.1



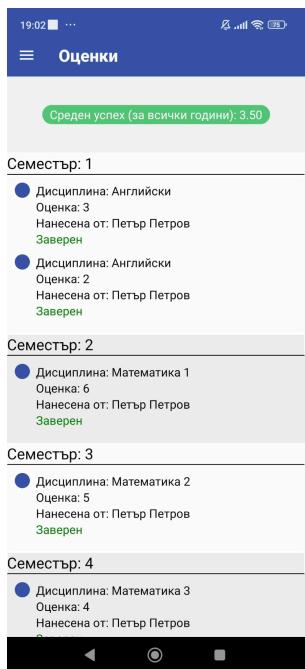
16.4. Оценки на студента

Страницата с оценки предоставя табличен преглед на всички оценки на студента, където се включват информация за дисциплината, оценяващия преподавател и статуса на завереност на предмета. Оценките са групирани по семестри.

В допълнение към конкретните оценки, страницата представя и средния успех на студента.

Интерфейсът е показан на фигура 16.4.1

Фиг 16.4.1

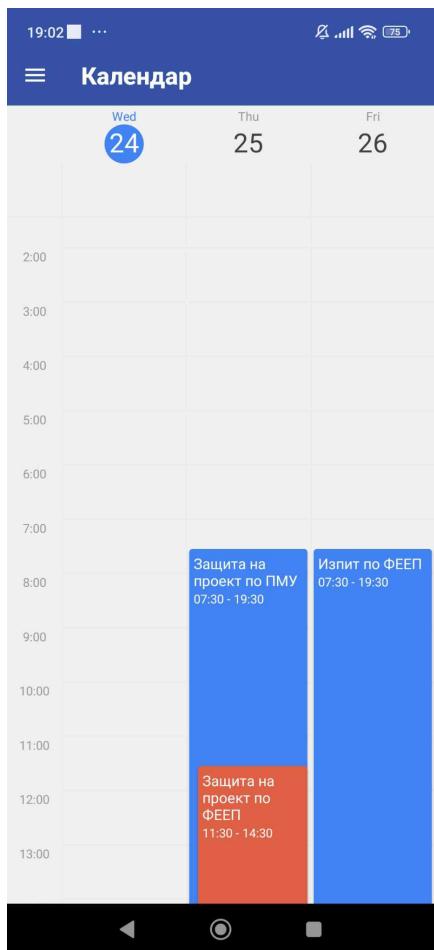


16.5. Календар на студента

На страницата за календар се представя графичен календар, върху който са отбелязани всички събития, свързани с учебния процес на студента.

Интерфейсът е показан на фигура 16.5.1

Фиг. 16.5.1

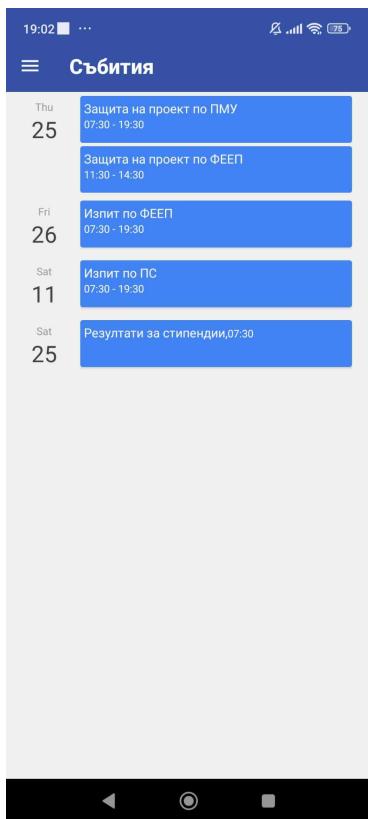


16.6. Събития на студента

Страницата за събития показва предстоящите събития на студента в списъчен вид

Интерфейсът е показан на фигура 16.6.1

Фиг 16.6.1



16.7. Здравно осигуряване на студента

Страницата за здравно осигуряване показва статуса на вноските за здравно осигуряване на студента - година, месец, осигурителна сума, осигуровка.

Интерфейсът е показан на фигури 16.7.1

Фиг 16.7.1

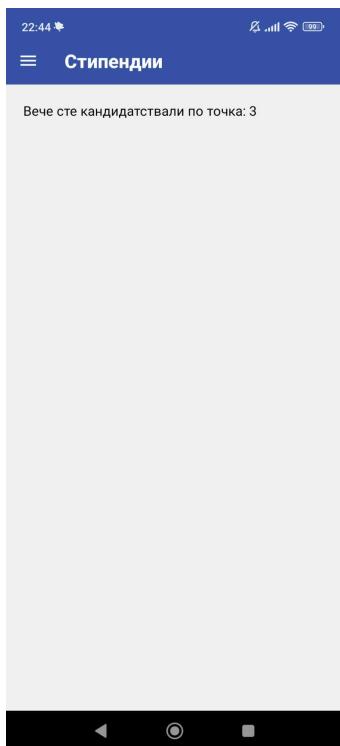
Година	Месец	Осиг. сума	Осигуровка
2024	1	886.35	70.91
2024	2	886.35	70.91
2024	3	702.00	48.28
2024	4	702.00	48.28
2024	5	702.00	48.28
2024	6	702.00	48.28
2024	7	639.00	56.16
2024	8	639.00	56.16
2024	9	639.00	56.16
2024	10	639.00	56.16
2024	11	520.00	56.16
2024	12	886.35	56.16
2023	1	520.00	70.91
2023	2	520.00	70.91

16.8. Стипендии, Общежития, Спорт на Студента

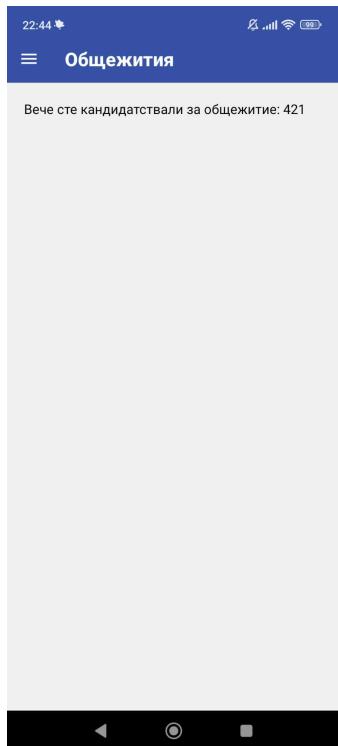
Страници, които дават кратка информация за статуса на стипендията на студента, статуса на общежитията му и спорта, на който той ходи.

Интерфейсът е показан на фигури 16.8.1, 16.8.2 и 16.8.3

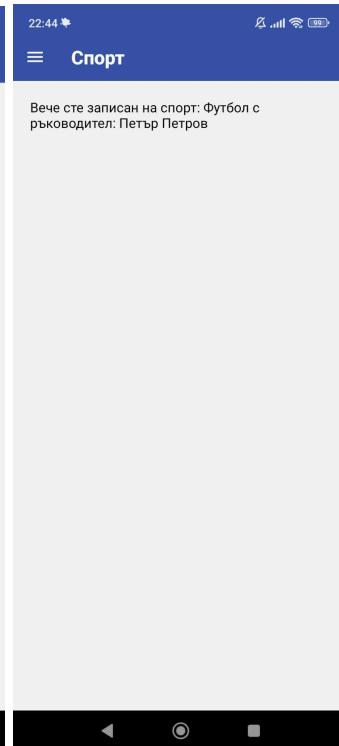
Фиг 16.8.1



Фиг 16.8.2



Фиг 16.8.1



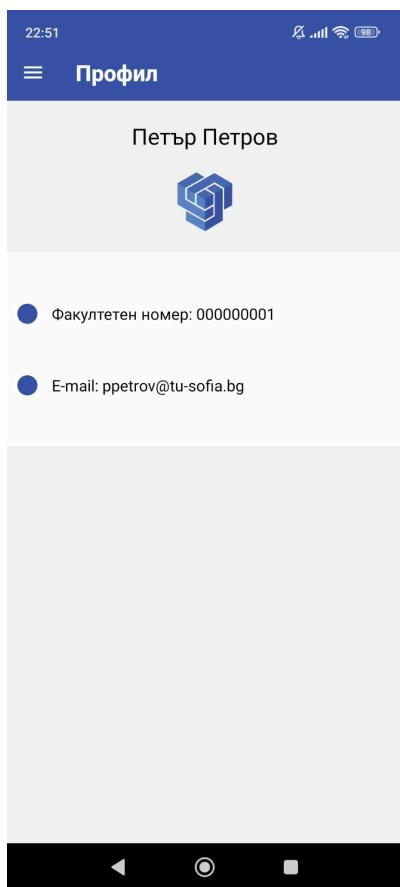
16.9. Информация за преподавател

След успешно влизане като преподавател, потребителя е посрещнат с информация за себе си.

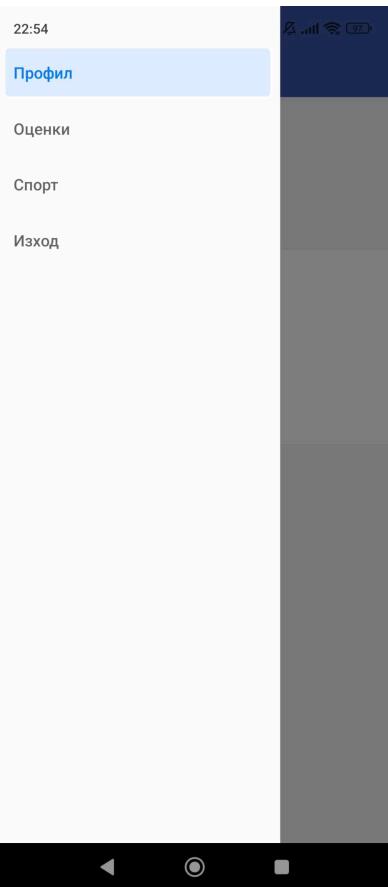
От тази страница той лесно може да се придвижи до останалите чрез навигацията тип чекмедже.

Интерфейсът е показан на фигури 16.9.1 и 16.9.2

Фиг 16.9.1



Фиг 16.9.1

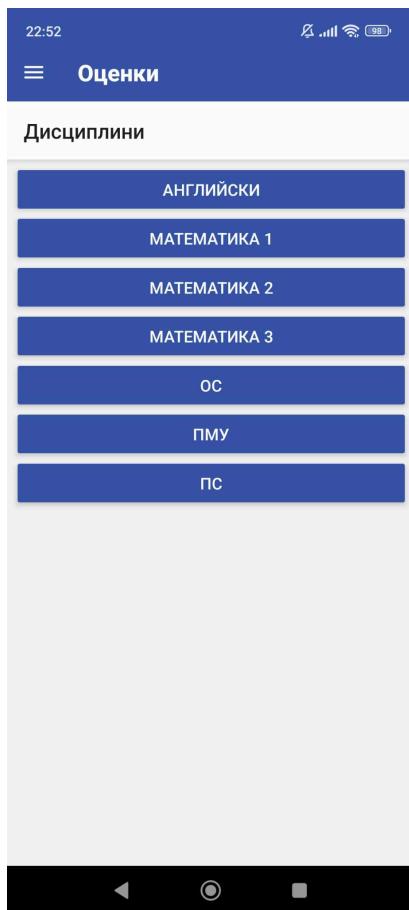


16.10. Екран за оценки - преподавател

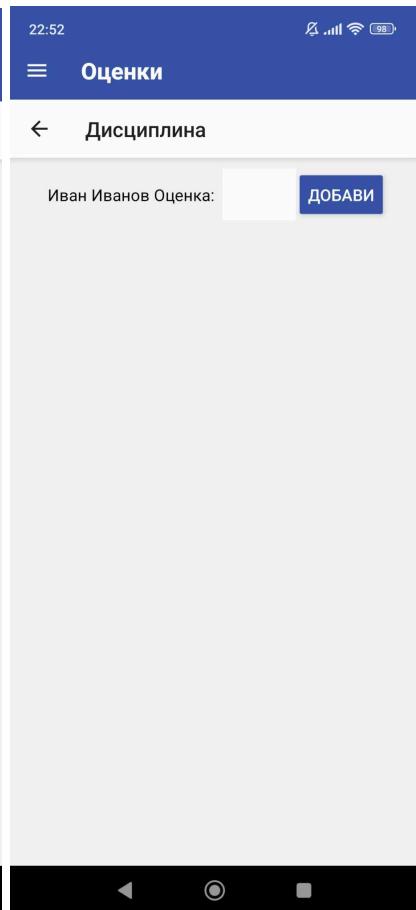
Екранът за оценки позволява на учителя да избере предмет и да нанесе оценка на студент. След като бъде нанесена оценка, студентът получава известие."

Интерфейсът е показан на фигури 16.10.1 и 16.10.2

Фиг. 16.10.1



Фиг. 16.10.2

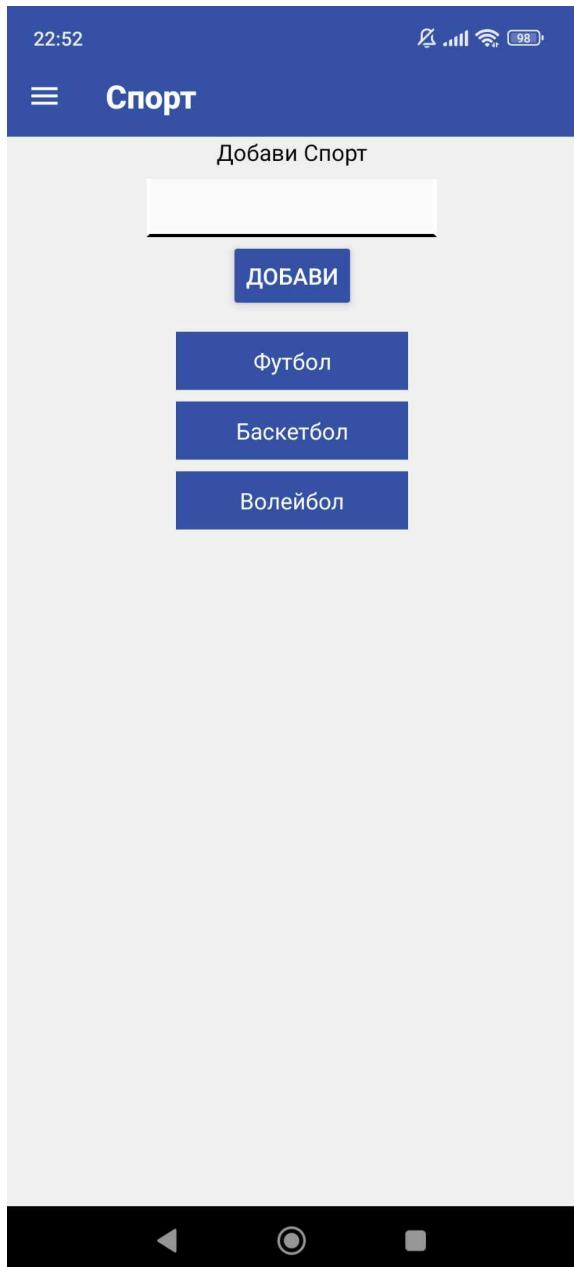


16.11. Екран спортове - преподавател

В екрана за спортове, учителя има възможност да създаде спорт

Интерфейсът е показан на фиг 16.11.1

Фиг. 16.11.1

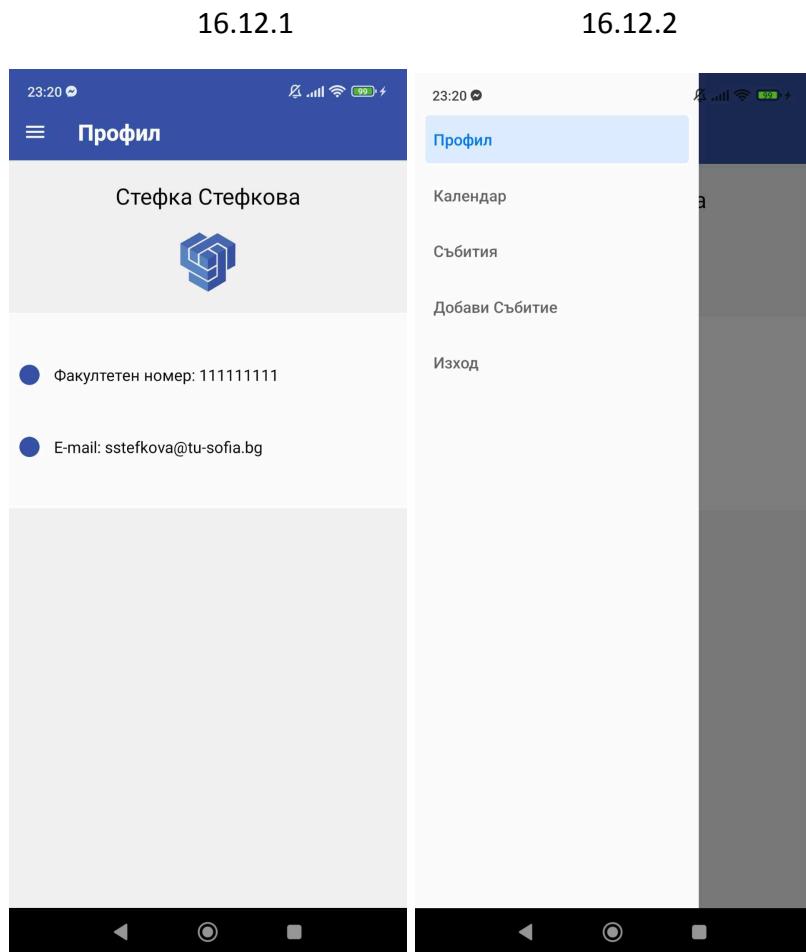


16.12. Информация за служител

След успешно влизане като служител, потребителя е посрещнат с информация за себе си.

От тази страница той лесно може да се придвижи до останалите чрез навигацията тип чекмедже.

Интерфейсът е показан на фигури 16.12.1 и 16.12.2

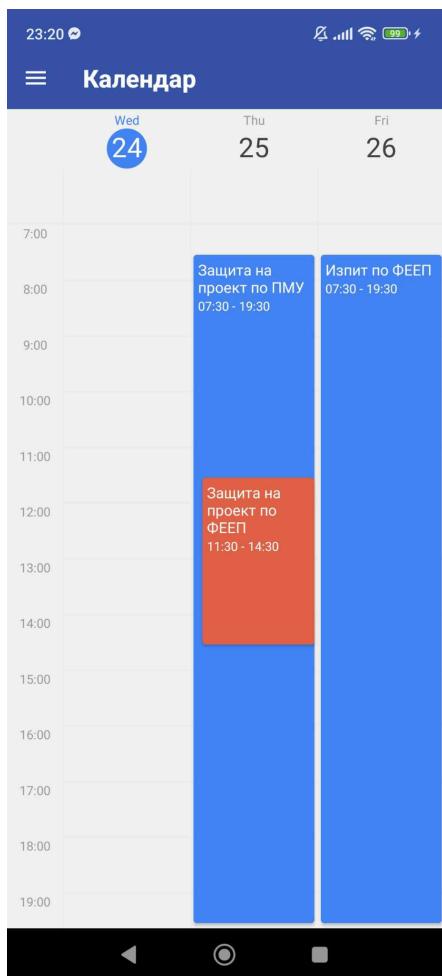


16.13. Календар на служителя

На страницата за календар се представя графичен календар, върху който са отбелязани всички събития на служителя

Интерфейсът е показан на фигура 16.13.1

Фиг. 16.13.1

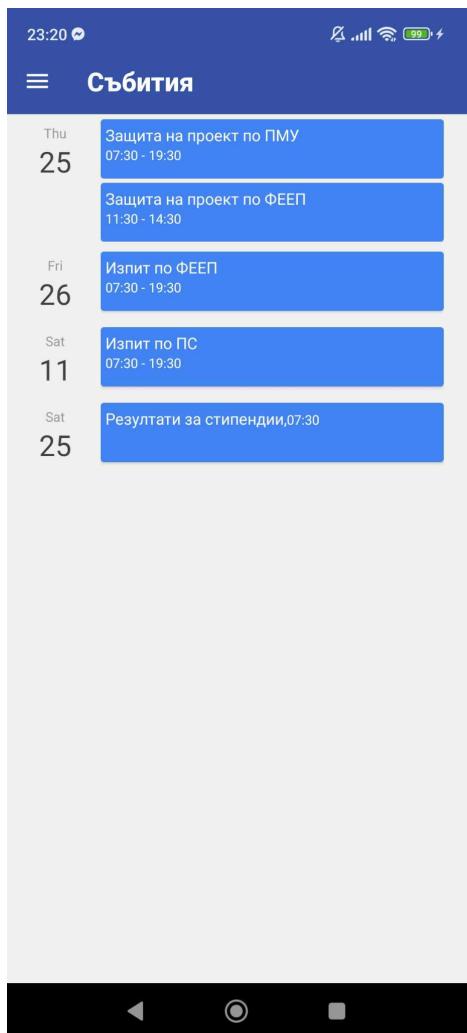


16.14. Събития на служителя

Страницата за събития показва предстоящите събития на служителя в списъчен вид

Интерфейсът е показан на фигура 16.14.1

Фиг. 16.14.1



16.15. Добави събития

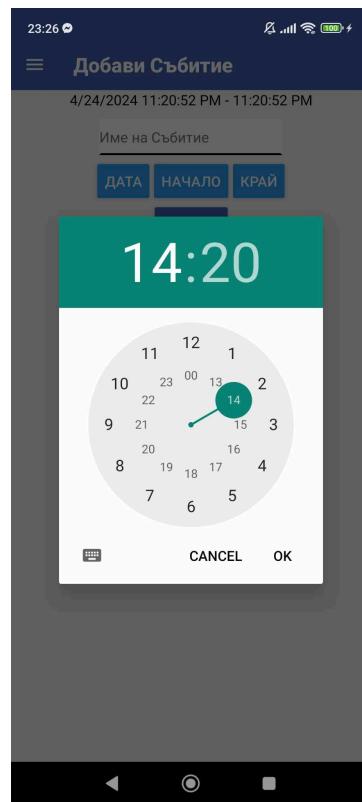
Страницата позволява на служителя да създава нови събития. Бутона за дата отваря календар, от който той може да избере датата на събитието. Бутона за начало отваря часовник, от който може да избере началния час на събитието. Бутона за край отваря часовник за край. Бутона 'Добави' създава ново събитие. След като събитието бъде създадено, до всяко събитие има бутон 'X', с който то може да бъде премахнато.

Интерфейсът е показан на фигураните от 16.15.1 до 16.15.2

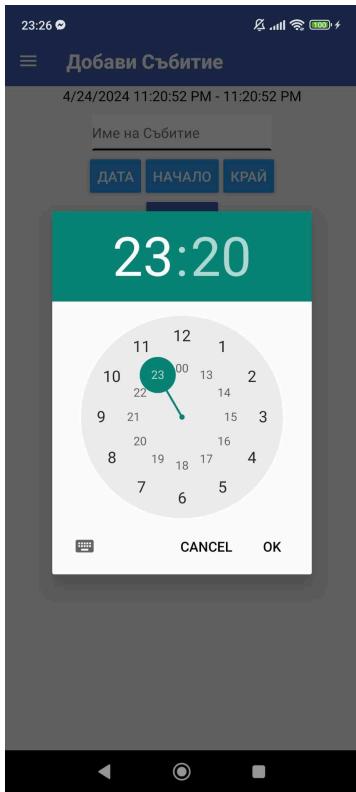
Фиг 16.15.1



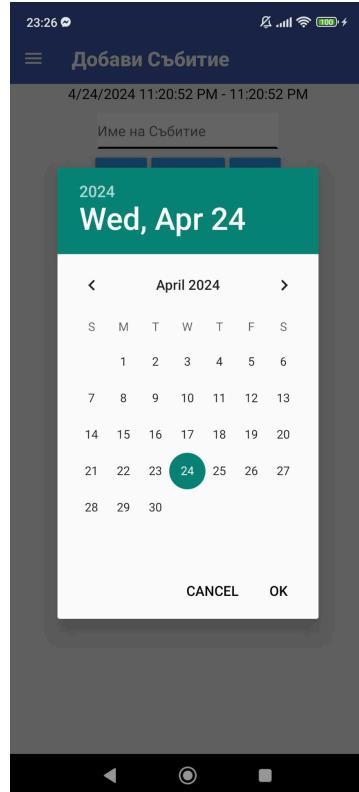
Фиг 16.15.2



фиг. 16.15.3



фиг. 16.15.4



17. Бъдещо развитие

17.1 . Интеграция със системи за управление на учебни материали

Интегриране на приложението със системи за управление на учебни материали (LMS), като например **Moodle** или **Canvas**, за улесняване на достъпа до курсове, материали и задачи.

17.2. Подобрени социални функции

Разширяване на социалните функции на приложението, като създаване на общности, форуми за дискусии, групови проекти и събития

17.3. Система за стажове и работни места за студентите

Създаване на платформа, където студентите могат да намират информация за стажове, работни места и възможности за кариерно развитие, както и да кандидатстват за тях директно през приложението.

17.4 Интерактивни карти на кампуса

Добавяне на функционалност, която позволява на студентите да намират лесно и бързо различните сгради, аудитории и служби на кампуса.

17.5. Нощен режим

Към приложението може да бъде добавена нощна черна тема, което е предпочитано от студентите поради няколко причини. Такава тема би подобрila

удобството при използване в нощи условия, би подчертала визулните елементи и би подобрila енергийната ефективност на устройството.

18. Заключение

Разработката на приложението представлява важна стъпка към модернизацията и подобряването на учебния процес и студентския живот. Чрез приложението, студентите получават лесен и удобен достъп до различни учебни ресурси, календар на събитията, оценки и актуална информация за кампуса. В същото време, преподавателите се възползват от възможността за по-ефективна комуникация и управление на учебните си материали.

Въпреки че разработването на приложението е завършило, има възможности за бъдещо развитие и подобрения. Включването на допълнителни функционалности като виртуални учебни материали, онлайн обучение, интерактивни карти на кампуса и социални функции може да подобри още повече потребителския опит и удовлетворение от приложението.

В заключение, мобилното приложение на университетската система представлява важен ресурс за студентите, преподавателите и администраторите, като допринася за по-ефективна и съвременна образователна среда.

19. Използвана литература

19.1. OPNsense

Документация на OPNsens - <https://docs.opnsense.org/index.html>

19.2. React Native

Документация на React Native - <https://reactnative.dev/docs/getting-started>

19.3. Laravel

Документация на Laravel - <https://laravel.com/docs/4.2/introduction>

19.4. Mysql

Документация Mysql - <https://dev.mysql.com/doc/>