Проект „НИКА“

**2025**



ПГЕЕ – гр. Банско

12/10/2025



Съдържание

[Увод 2](#_Toc211187153)

[Контекст и история 3](#_Toc211187154)

[Какво печелим всички (стойност и ползи) 3](#_Toc211187155)

[За организаторите на състезания 3](#_Toc211187156)

[За учениците 4](#_Toc211187157)

[За местната общност 4](#_Toc211187158)

[Какво може системата (функционалности на разбираем език) 4](#_Toc211187159)

[Архитектура на решението 5](#_Toc211187160)

[Хардуерен слой: надеждно засичане и визуално доказателство 5](#_Toc211187161)

[Софтуерен междинен слой: „роботът“, който свързва хардуера с системата 7](#_Toc211187162)

[Backend: стабилен „гръбнак“ с Django и DRF 9](#_Toc211187163)

[Какво прави backend-ът накратко 9](#_Toc211187164)

[Основни модели данни 9](#_Toc211187165)

[Примерни дефиниции от models.py 9](#_Toc211187166)

[Сериализатори: чист интерфейс към данните 10](#_Toc211187167)

[REST endpoints: как „роботът“ и UI говорят с backend-а 11](#_Toc211187168)

[Примерни view класове (DRF) 11](#_Toc211187169)

[Frontend: един Vue шаблон (SDI), живи данни и операторски контрол 12](#_Toc211187170)

[Ключови елементи от имплементацията 14](#_Toc211187171)

[Синхронизация: веднага след събитие, навсякъде 16](#_Toc211187172)

[Как работи накратко 16](#_Toc211187173)

[Защо е ефективно 17](#_Toc211187174)

[Заключение 17](#_Toc211187175)

# Увод

НИКА е родена от нуждата състезанията да се провеждат спокойно и уверено, без излишна техника и стрес по финала. Представете си екип, който пристига на терен, сглобява всичко за минути и започва работа офлайн — без да чака интернет, без да настройва сложни системи. IR бариерата засича момента на пресичане с точност, камерите дават ясна картина кой е минал и кога, а операторът има ясен екран, на който всичко се подрежда само. Това е идеята на НИКА: компактно решение, което събира в себе си надежден хардуер, лек софтуер и ясен интерфейс, и ги кара да работят заедно като добре настроен екип.

Проблемът, който решаваме, е до болка познат на организаторите: как да измериш време прецизно и проверимо, как да свържеш резултата с конкретния състезател, и как да дадеш на публиката и съдиите увереност, че класирането е честно. В динамиката на масови бягания, планински трасета и променливо време, грешките се „раждат“ лесно — скъсани номера, кал, плътни групи на финала. НИКА комбинира автоматично разпознаване на номера от камера с IR тригер и фотофиниш кадри, така че зад всяко време да стои визуално доказателство и прозрачна история. Там, където автоматиката срещне нещо необичайно, операторът има възможност да коригира бързо и без драматизъм.

Системата е създадена за хората, които правят събитията възможни: организатори на бягания и велонадпревари, клубове, общини и доброволци, които искат да работят с инструменти, а не с препятствия. За тях изчистихме интерфейса до един екран, в който всичко живее — регистрация, старт, „живото“ състезание, фотофиниш и класиране. Няма разхвърляни прозорци и сложни менюта; само ясни режими и данни, които се обновяват сами през секунди.

Под капака НИКА остава прагматична. Хардуерният слой е прост и надежден: индустриална IR бариера, Arduino и две IP камери с PoE — това е достатъчно, за да имате точка на истината и добра картина на финала. Софтуерният „робот“ е лек Python сървис, който без умора слуша IR сигнали и видеопотоци, разпознава номера, маркира финишите и изпраща всичко важно към бекенда. Django с REST API пази данните подредени и достъпни, а Vue интерфейсът ги показва така, че операторът да се чувства в контрол. Синхронизацията е умишлено проста: кратки периодични заявки, които гарантират, че всички виждат промените почти веднага след събитие — без излишна инфраструктура.

Днес НИКА е работещ прототип. Системата вече е минала през реално събитие — с истински финали, истинска кал и истинско напрежение — и се държа така, както очаквахме: тихо, последователно и надеждно. Екипът около проекта расте, а с него и плановете: по-точен OCR при трудни условия, по-богати справки за организаторите, по-лесен импорт на списъци и още малки удобства, които правят голяма разлика в деня на състезанието.

В крайна сметка НИКА е обещание за спокойствие: бърза подготовка, надеждно засичане, автоматично класиране и готови документи — всичко нужно, за да се концентрирате върху събитието и хората, а не върху техниката.

# Контекст и история

Поводът за проекта е конкретна местна нужда. Организаторите от местно спортно сдружение „АлПиринСки“ подготвяха планинското състезание „Тодорка Vertical“ и търсеха надежден, достъпен и мобилен начин за точно времеизмерване и бързо изкарване на официални класирания. Стандартните решения са скъпи, трудни за адаптиране към различни трасета и често изискват външни екипи. В този контекст се роди идеята за система, която да съчетава прецизно времеизмерване, фотофиниш и автоматично разпознаване на състезателни номера – с възможност за работа офлайн и лесно пренасяне на терен.

Началото беше поставено в ПГЕЕ – Банско като вътрешна инициатива, ръководена от инж. Георги Бориков, с активното участие на ученици и подкрепата на директора инж. Тодорова. Ключов принос имаше ученикът от 12-ти клас Виктор Василев, който обедини знания по програмиране, изкуствен интелект и електроника, за да създаде първия работещ прототип. Екипът заложи на практично решение: интеграция на IR бариера за прецизен момент на финиша, камерa за фотофиниш и компютърно зрение за разпознаване на състезателни номера, управлявани от уеб интерфейс, удобен за оператори.

Развитието бе естествено и ускорено от реалната практика. Първата версия успешно беше изпробвана при провеждането на „Тодорка Vertical“, което повиши интереса сред учениците и привлече нови участници в екипа. Така проектът прерасна в ядро на клуб по „младежко техническо творчество“, където ученици от 8. до 12. клас работят в специализирани подекипи – софтуер, ИИ, хардуер и тестване. Днес системата се развива по план: стабилизира се разпознаването, подобрява се операторският интерфейс и се добавят нови функционалности, а партньорството с местните спортни организации и общината дава рамка за устойчивост и по-широко приложение.

# Какво печелим всички (стойност и ползи)

## За организаторите на състезания

* Точност и надеждност: IR бариера и фотофиниш гарантират коректно време и доказателствен кадър при всяко пресичане на финала.
* По-бързо класиране: системата автоматично подрежда резултатите по категории и генерира готови документи за минути.
* По-добра логистика на терен: работи офлайн, мобилна е и се настройва бързо за различни трасета; позволява ръчен и автоматичен режим паралелно.
* Прозрачност и спокойствие: при спорни случаи има визуално доказателство; операторският интерфейс позволява лесни корекции и бележки.

## За учениците

* Реален, смислен проект: работят по система, която се използва на истински събития и носи отговорност за резултатите.
* Интегрирани умения: съчетават софтуер (уеб и бекенд), изкуствен интелект (компютърно зрение) и хардуер (сензори, микроконтролери).
* Екипна работа и роли: организация на подекипи (софтуер, ИИ, хардуер, тестове), планиране, документация и публични демонстрации.
* Видимост и развитие: участие в местни и национални изложения и състезания, портфолио от реални решения и контакти с партньори.

## За местната общност

* Местна иновация: Банско развива собствено технологично решение, пригодено за планински и масови спортни събития.
* Повече видимост: модерно времеизмерване и бързи класирания подобряват имиджа на състезанията и привличат участници и медии.
* Партньорство „училище–спорт–община“: практичен пример как образованието подкрепя местни инициативи и създава стойност за всички.
* Устойчивост и наследство: изграден капацитет в ученици и учители, който остава в региона и се надгражда с всяко следващо събитие.

## Какво може системата (функционалности на разбираем език)

Системата е замислена така, че да обслужва реални състезания без излишна сложност за организаторите и операторите. Тя комбинира автоматизация там, където е най-полезна, и дава контрол на човека, когато е нужно.

* Ръчен и автоматичен режим, които работят заедно
  + При нормална работа системата засича финиша автоматично чрез IR бариера и камери. Ако в даден момент има спорен случай или нестандартна ситуация, операторът може да отбележи ръчно време и да добави бележка. Двата режима не си пречат – автоматиката върши „тежката работа“, а ръчният режим е резервен план и инструмент за прецизиране.
* Офлайн работа и мобилност на терен
  + Цялата система може да работи на едно място – лаптоп, камери и IR бариера. Няма зависимост от интернет: резултатите се записват локално и, при нужда, се синхронизират след събитието. Оборудването е преносимо и се сглобява бързо, което е важно за планински трасета и динамични условия.
* Пълна подготовка преди старта
  + Организаторите могат да въведат или импортират списък на състезателите, да оформят стартови листи по групи и категории, и да отбележат различни състояния – например „старт“ или „нестартирал“. Така още преди първия сигнал всички данни са подредени и готови.
* Разпознаване на състезателни номера и пресичане на финала
  + Камерата при финала „чете“ състезателните номера (bib) и ги свързва с конкретните участници. Едновременно с това IR бариерата дава точния момент на пресичане. Комбинацията от двете осигурява както точност, така и надеждна идентификация, дори при по-висока плътност на финиширащите.
* Фотофиниш и точност до 0.1 секунда
  + За всяко засечено пресичане системата запазва кадър (или кратка серия) от финалната камера. Това е „визуалното доказателство“ при оспорвани резултати. Времето се измерва с точност до 0.1 секунди, което е достатъчно за повечето масови и планински състезания.
* Класиране автоматично, с възможност за корекции
  + В реално време се оформят временни класирания по групи и категории. Ако операторът забележи несъответствие (например объркан номер при кално време или скъсана цифра), може да коригира асоциирането ръчно. Системата пази следи от измененията за прозрачност.
* Готови официални документи
  + След финала класирането се генерира автоматично във формат MS Word: със заглавие на състезанието, категория, позиции, времена и допълнителни бележки. Документите са готови за печат и подпис, без допълнително форматиране.
* История и справки за много събития
  + Не се работи „на чисто“ всеки път. Системата пази архив на състезанията, участниците и резултатите. Организаторите могат да правят справки – например за участие на клубове, средни времена по категории или сравнение между издания на едно и също събитие.

С две думи, системата съчетава удобство за операторите, прозрачност за състезателите и сигурност за организаторите: бърза подготовка, надеждно засичане, лесни корекции и готови документи – всичко, необходимо за спокойно и професионално провеждане на състезание.

# Архитектура на решението

## Хардуерен слой: надеждно засичане и визуално доказателство

Хардуерът е проектиран да е прост за сглобяване на терен, устойчив на смущения и достатъчно прецизен за масови и планински събития. Комбинираме индустриален инфрачервен лъч за точно тригериране на финала с две IP камери – едната за разпознаване на номера, другата за фотофиниш и броене на преминавания. Всичко се свързва към лаптоп през стандартни интерфейси, без специализирани контролери.

* IR бариера с Arduino адаптер → сериен интерфейс към компютър
  + Бариера: използваме готов комплект за периметрова охрана Photoelectric beam detector ALX-60 (2 beams, до 60 m). Лъчът е тесен (≈2°) и модулиран, което минимизира фалшиви сработвания от слънце, отражения и др. Релейният изход е удобен за интеграция, а минималното време за реакция (настройваемо, от порядъка на наносекунди) е далеч над нуждите ни за спортно времеизмерване.
  + Свързване: релето на бариерата (NC – нормално затворен контакт) се връзва към пин D2 на Arduino Nano. Arduino се захранва през USB от лаптопа и използва вграден INPUT\_PULLUP, за да „чете“ ясни логически нива без външни резистори.
  + Логика: при прекъсване/възстановяване на лъча Arduino изпраща четим текстов сигнал по сериен порт (USB → виртуален COM) към софтуера на лаптопа. Добавен е кратък debounce, за да отсеем механични трептения.
  + Примерен Arduino sketch (използван в системата):

const int BARRIER\_PIN = *2*;  
int lastState = HIGH; *// При PULLUP - HIGH = незадействано*void setup() {  
 pinMode(BARRIER\_PIN, INPUT\_PULLUP);   
 Serial.begin(*9600*);  
}  
  
void loop() {  
 int currentState = digitalRead(BARRIER\_PIN);  
  
 *// Преход от НЕ задействана (HIGH) към задействана (LOW)* if (lastState == HIGH && currentState == LOW) {  
 Serial.println("DEACTIVATED"); *// Бариера освободена* delay(*20*); *// debounce* }  
  
 *// Преход от задействана (LOW) към НЕ задействана (HIGH)* if (lastState == LOW && currentState == HIGH) {  
 Serial.println("ACTIVATED"); *// Бариера задействана* delay(*20*); *// debounce* }  
  
 lastState = currentState;  
 delay(*2*); *// Main loop debounce*}

Как се интерпретира: „ACTIVATED“ маркира събитие на финиша (пресичане), „DEACTIVATED“ – освобождаване. Софтуерният слой взима UTC време при „ACTIVATED“ и го съхранява за класиране, като по избор асоциира и кадър от камера.

* Две IP камери за визуална идентификация и фотофиниш
  + Модел: Dahua DH-IPC-HFW1239TL1-A-IL (2MP, 2.8 mm). Избрани заради добра светлочувствителност, стабилен RTSP поток и надеждност на открито.
  + Роля на камерите:
    - Камера 1 „Разпознаване“: ориентирана така, че състезателните номера (bib) да заемат предвидимо място в кадъра при доближаване/пресичане. Потокът се обработва локално с модели за детекция (YOLO) и OCR (EasyOCR), за да се извлече номерът и да се свърже с IR събитието.
    - Камера 2 „Фотофиниш и броене“: фокусирана върху линията на финала. При IR тригер се запазва кадър/серия за доказателство и за ръчна верификация при спор. Служи и за допълнително броене при плътни финиши.
  + Свързаност и захранване:
    - Всички устройства са в една локална мрежа със статични IP адреси. Използваме PoE switch Dahua DH-PF3006-4ET-60, който комбинира мрежа и захранване в един кабел за всяка камера. Това опростява монтажа на терен, намалява нуждата от допълнителни адаптери и повишава надеждността.
    - Поток: достъп по ONVIF/RTSP от локалния лаптоп; не е необходим интернет. Записът на кадри става локално, а при нужда – се архивира след събитието.
* Предимства на тази комбинация
  + Прецизност и устойчивост: хардуерният IR тригер дава точен момент на финиша; модулираният лъч и тесният ъгъл ограничават смущения. Камерите осигуряват визуална верификация и идентификация.
  + Простота и сервизопригодност: стандартен Arduino Nano, стандартен сериен порт, стандартни IP камери с PoE. При проблем компонентите се заменят лесно, без специализирани инструменти.
  + Мобилност: един PoE суич, два Ethernet кабела, USB към лаптоп – готово за монтаж за минути. Работи изцяло офлайн.

С тази хардуерна основа системата постига точно времеизмерване, надеждна идентификация и проверими резултати, дори в динамична среда и при голям поток финиширащи.

## Софтуерен междинен слой: „роботът“, който свързва хардуера с системата

Тук работи нашият „робот“ – лек Python сървис, който непрекъснато „слуша“ IR бариерата и камерите за събития и автоматично изпраща време, изображения и метаданни към backend-а. Идеята е проста: роботът наблюдава, разпознава и отчита вместо оператор, като подава чисти и своевременни данни към системата за класиране.

Как работи накратко

* Свързване и настройки: роботът чете конфигурация (IP адреси, RTSP параметри, сървър, токен) и изгражда RTSP адресите към камерите:

def make\_rtsp\_url(cam, cfg):  
 ip = cfg[f'camera\_{cam}']['ip']  
 port = cfg[f'camera\_{cam}']['port']  
 username = cfg[f'camera\_{cam}']['username']  
 pwd = cfg[f'camera\_{cam}']['password']  
 channel = cfg[f'camera\_{cam}']['channel']  
 subtype = cfg[f'camera\_{cam}']['subtype']  
 return f'rtsp://{username}:{pwd}@{ip}:{port}/cam/realmonitor?channel={channel}&subtype={subtype}'

* Връзка с backend-а: модулът APIclient осигурява сигурна комуникация (Token) и периодична синхронизация на параметрите на състезанието (начален час, статус, стартов списък):

race\_api = RaceClient(server\_url, server\_token)  
race\_api.load\_sys\_params()  
  
race\_api.periodic\_sync(sync\_interval=1.0)

* Наблюдение на финалната линия: роботът „гледа“ RTSP потока, детектира хора с YOLO, проследява ги със SORT и отчита преминаване през виртуална линия на финала.

results = model(frame)

# YOLO детекция

tracks = tracker.update(dets\_for\_sort)

# SORT проследяване

if last\_side < 0 and side > 0:

# преминаване през линията

crossed\_counter += 1

ath = race\_api.mark\_first\_as\_finished(...)

* Фотофиниш и доказателство: при засечено преминаване се записва кадър и се прикачва към съответния състезател в бекенда.

with tempfile.NamedTemporaryFile(suffix=".jpg", delete=False) as tmp\_file:  
 tmp\_filename = tmp\_file.name  
  
try:  
 *# Записвам снимката (файлът вече е затворен и не е lock-нат)* cv2.imwrite(tmp\_filename, frame)  
 for athlete\_id in ath:  
 url = f"{server\_url}/api/athletes/{athlete\_id}/add\_photo/"  
 with open(tmp\_filename, 'rb') as img:  
 files = {'image': img}  
 response = requests.post(url, files=files)

* Времева синхронизация: роботът изчислява локално време спрямо сървъра, за да запише резултати с точност до десети.

def format\_timer(self, ofs=0):  
 seconds = ((int(time.time() \* 1000) - self.time\_offset) - self.start\_time) / 1000 +0.1\*ofs  
 total\_seconds = int(seconds)  
 deci = int((seconds - total\_seconds) \* 10) *# десети* hours = total\_seconds // 3600  
 minutes = (total\_seconds % 3600) // 60  
 secs = total\_seconds % 60  
  
 *# ЧЧ:Без водеща нула, ММ:СС задължително по две цифри, десети след точка* return f'{hours}:{minutes:02}:{secs:02}.{deci}'

Защо този подход е надежден

* Работи локално и офлайн: RTSP и сериен интерфейс се четат на място; данните се изпращат към локален сървър.
* Реагира в реално време: детекция, проследяване и маркиране на финиши стават в рамките на кадри.
* Осигурява прозрачност: към всяко време има снимка-доказателство и проследимост кой/кога е маркиран като финиширал.

С две думи, „роботът“ е автоматичният оператор на терен: настройва се веднъж, след което неуморно приема сигнали, анализира видео, маркира финиши и подава чисти данни към системата – бързо, последователно и без излишни човешки действия.

## Backend: стабилен „гръбнак“ с Django и DRF

Backend-ът е изграден с Django и Django REST Framework (DRF) и играе ролята на надежден посредник между „робота“, операторския интерфейс и базата данни. Той приема събитията от терена, съхранява структурирано информацията и я свързва със състезанието и състезателите, така че класирането да е винаги последователно, проследимо и готово за показване.

### Какво прави backend-ът накратко

* Управлява системни параметри на състезанието: име, статус, начало, текущ пореден номер при финиш.
* Поддържа групи/категории, списък на състезатели и техните статуси.
* Приема финишни събития и снимки-доказателства (качени от „робота“), свързани към конкретни състезатели.
* Осигурява API за стартиране/спиране, синхронизация на време, сортиране/преглед и корекции.

### Основни модели данни

* Competition: държи статуса, началния час и next\_num (брояч за финиширащи).
* Group: категории/групи за класиране.
* Athlete: име, стартов номер (bib), резултатно време, статус, пол и група.
* AthletePhoto: снимки към конкретен състезател (за фотофиниш/верификация).

### Примерни дефиниции от models.py

class Competition(models.Model):  
 name = models.CharField('Име на състезанието', max\_length=100)  
 status = models.PositiveSmallIntegerField('Текущ статус на състезанието', null=True, default=0)  
 start\_time = models.DateTimeField('Начало на състезанието', null=True, blank=True) *# UTC!* next\_num = models.PositiveSmallIntegerField('Текущ пореден номер на пристигащ състезател', null=True, default=0)  
  
class Group(models.Model):  
  
 name = models.CharField('Име нагрупата', max\_length=30, default='')  
 comment = models.CharField('коментар/пояснение', max\_length=80, default='', blank=True)

class Athlete(models.Model):  
 name = models.CharField('Име на състезателя', max\_length=80)  
 bib\_number = models.IntegerField('Стартов номер', default=0)  
 result\_time = models.CharField('Време', max\_length=10, default='0:00:00.0')  
 num = models.PositiveSmallIntegerField('Текущ пореден номер на пристигащ състезател', null=True, default=999)  
 status = models.SmallIntegerField('Статус на състезателя',  
 choices=[(-1, 'не е стартирал'), (0, 'дисквалифициран'), (1, 'регистриран'), (2, 'финиширащ'),  
 (3, 'финиширал')], default=1)  
 group = models.ForeignKey(  
 Group,  
 on\_delete=models.PROTECT, *# За да не се позволи изтриване на ползвана група* related\_name='athletes',  
 verbose\_name='Група/категория'  
 )  
 user = models.CharField('Създадено от', max\_length=1, default='М')  
 gender = models.BooleanField('Пол', default=True, choices=[(True, 'мъж'), (False, 'жена'), ])  
  
class AthletePhoto(models.Model):  
 athlete = models.ForeignKey(  
 Athlete,  
 related\_name="photos",  
 on\_delete=models.CASCADE  
 )  
 image = models.ImageField(upload\_to="athlete\_photos/")

### Сериализатори: чист интерфейс към данните

* Осигуряват четим и предвидим JSON за фронтенда и „робота“. Пример:

class AthleteSerializer(serializers.ModelSerializer):  
 group = GroupSerializer(read\_only=True)  
 group\_id = serializers.PrimaryKeyRelatedField(  
 queryset=Group.objects.all(),  
 source='group',  
 write\_only=True  
 )  
  
 class Meta:  
 model = Athlete  
 fields = ['id', 'name', 'bib\_number', 'result\_time', 'num', 'status', 'group', 'group\_id', 'user', 'gender']

### REST endpoints: как „роботът“ и UI говорят с backend-а

* Системни параметри и групи:
  + GET/POST /api/sysparams/
  + GET/POST /api/groups/
* Състезатели:
  + GET/POST /api/athletes/
  + GET/PUT/PATCH/DELETE /api/athletes//
  + GET /api/athletes/sort// (подреждане по bib, num, result\_time)
  + POST /api/athletes/bulk-num-update/ (бърза размяна/корекция на num)
  + POST /api/athletes/disqualify/ (масово маркиране на DQ по правило)
* Фотофиниш:
  + POST /api/athletes/<athlete\_id>/add\_photo/ (качване на снимка към състезател)
  + GET /api/athlete-photos/ (преглед на снимки)
* Управление на състезанието:
  + GET /api/competition/time/ (server\_time и start\_time за синхронизация)
  + POST /api/competition/start/ (отбелязване на начало)
  + PATCH /api/competition/status/ (промяна на статус)
  + POST /api/competition/nextnum/inc/ (инкремент на next\_num при финиш)

### Примерни view класове (DRF)

class AthleteListSortView(generics.ListCreateAPIView):  
 serializer\_class = AthleteSerializer  
  
 def get\_queryset(self):  
 sk = self.kwargs.get('sk')  
 order\_field = 'bib\_number'  
 if sk == 2:  
 order\_field = 'num'  
 elif sk == 3:  
 order\_field = 'result\_time'  
 return Athlete.objects.all().order\_by(order\_field)  
  
  
class AthleteDetailView(generics.RetrieveUpdateDestroyAPIView):  
 queryset = Athlete.objects.all()  
 serializer\_class = AthleteSerializer  
  
  
class CurrentStartTime(APIView):  
 def get(self, request):  
 comp = Competition.objects.get(id=1)  
 return Response({  
 'start\_time': comp.start\_time.isoformat() if comp.start\_time else None,  
 'server\_time': now().isoformat()  
 })

Защо този бекенд е надежден

* Ясна схема и стабилни връзки: всеки запис е свързан с група/състезание; снимките са обвързани с конкретен състезател.
* Прости, разбираеми API-та: фронтендът и „роботът“ работят с предвидими ендпойнти и формати.
* Проследимост и контрол: статуси, временни стойности и снимки-доказателства позволяват корекции и проверки при спорни случаи.

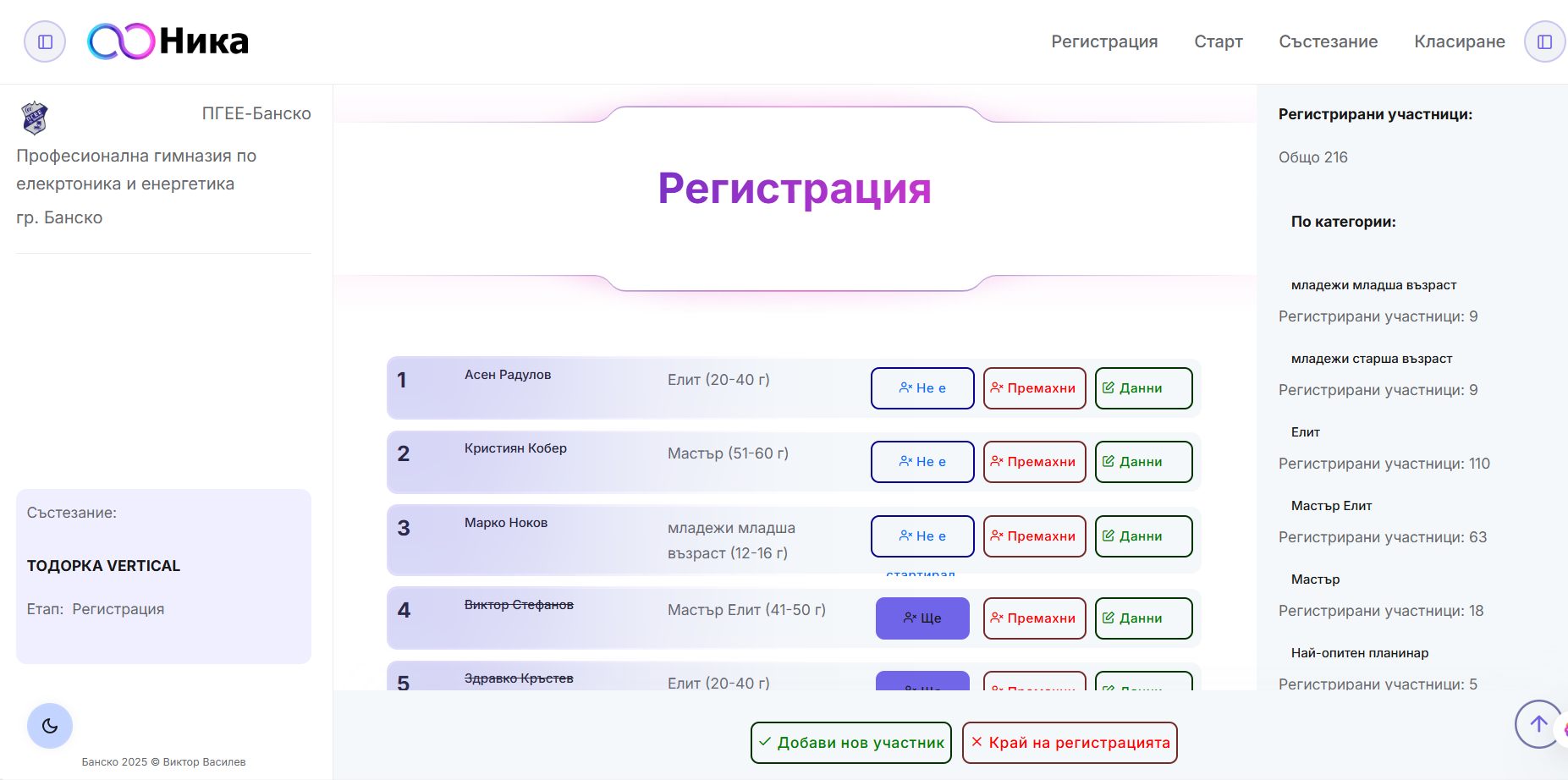
С две думи, бекендът е „гръбнакът“, който приема сигналите от терена, пази данните чисти и свързани, и ги предоставя веднага на интерфейса за класиране и отчет.

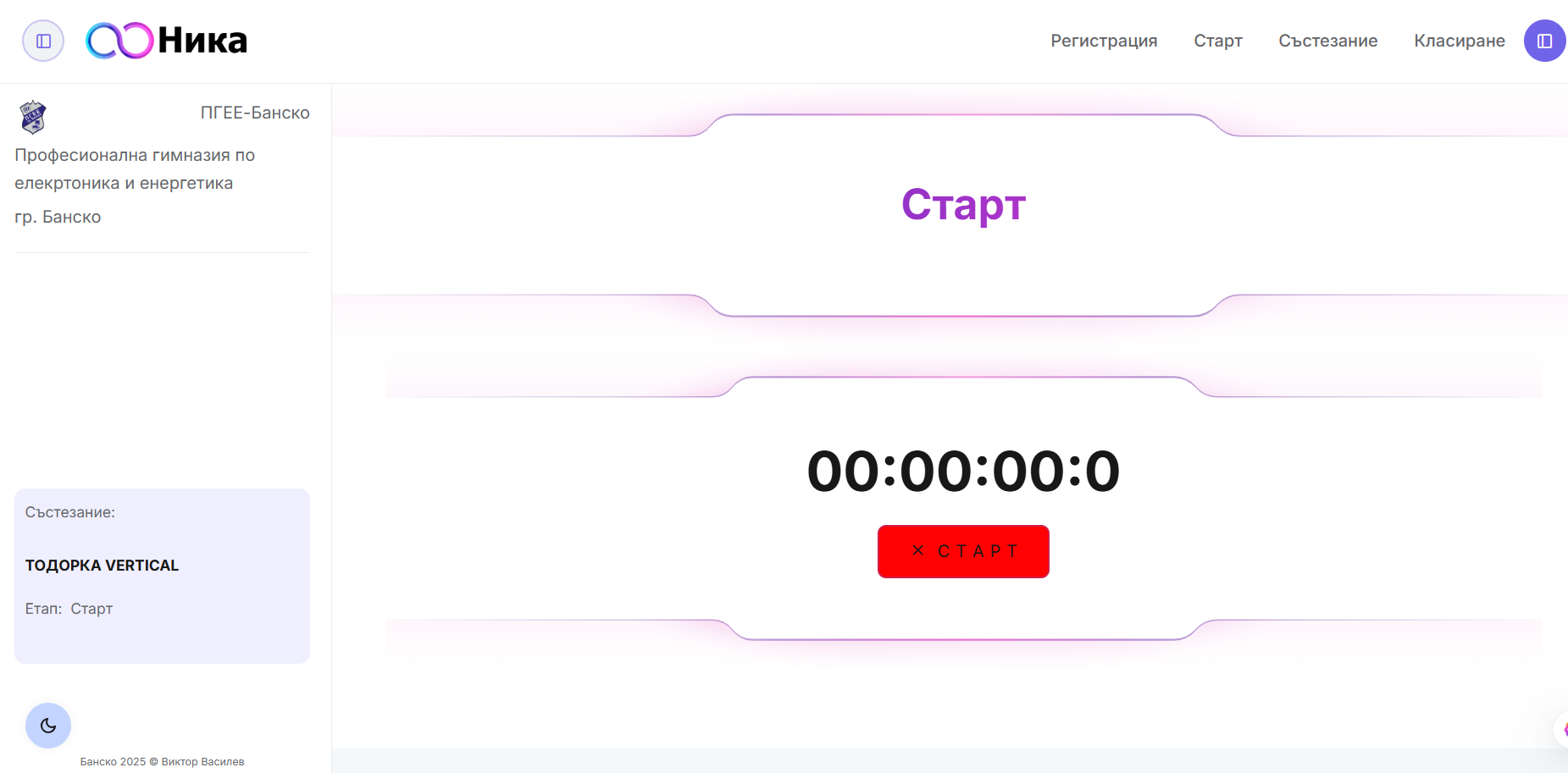
## Frontend: един Vue шаблон (SDI), живи данни и операторски контрол

Фронтендът е умишлено опростен и концентриран: почти цялата логика и изглед са в един HTML шаблон плюс един Vue скрипт. Така постигаме SDI-подобен интерфейс, където отделните секции се показват/скриват според етапа на състезанието чрез променливата showMode. Комуникацията с бекенда е през axios, а HTML базата е адаптиран готов темплейт, персонализиран за нуждите на времеизмерването.

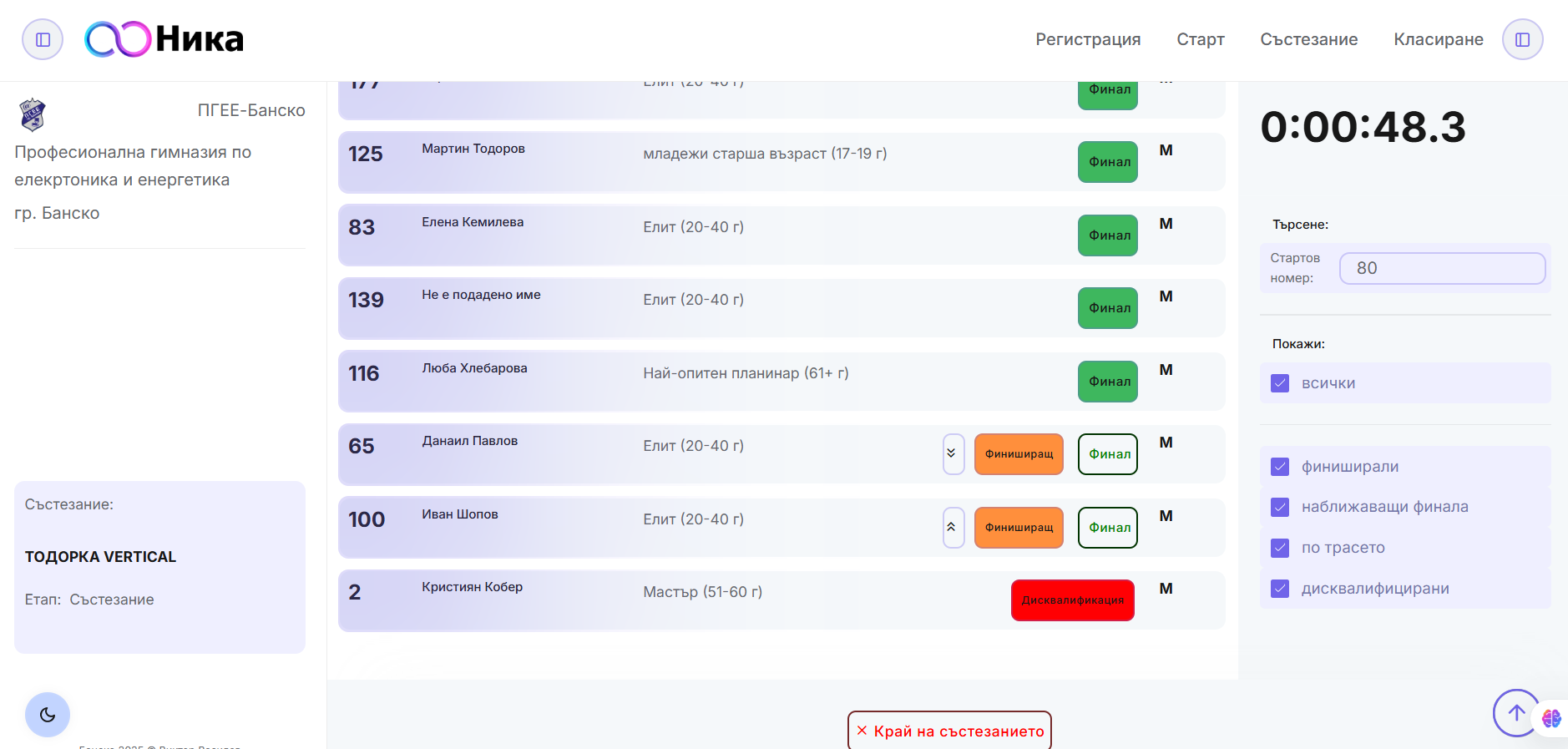
Как работи накратко

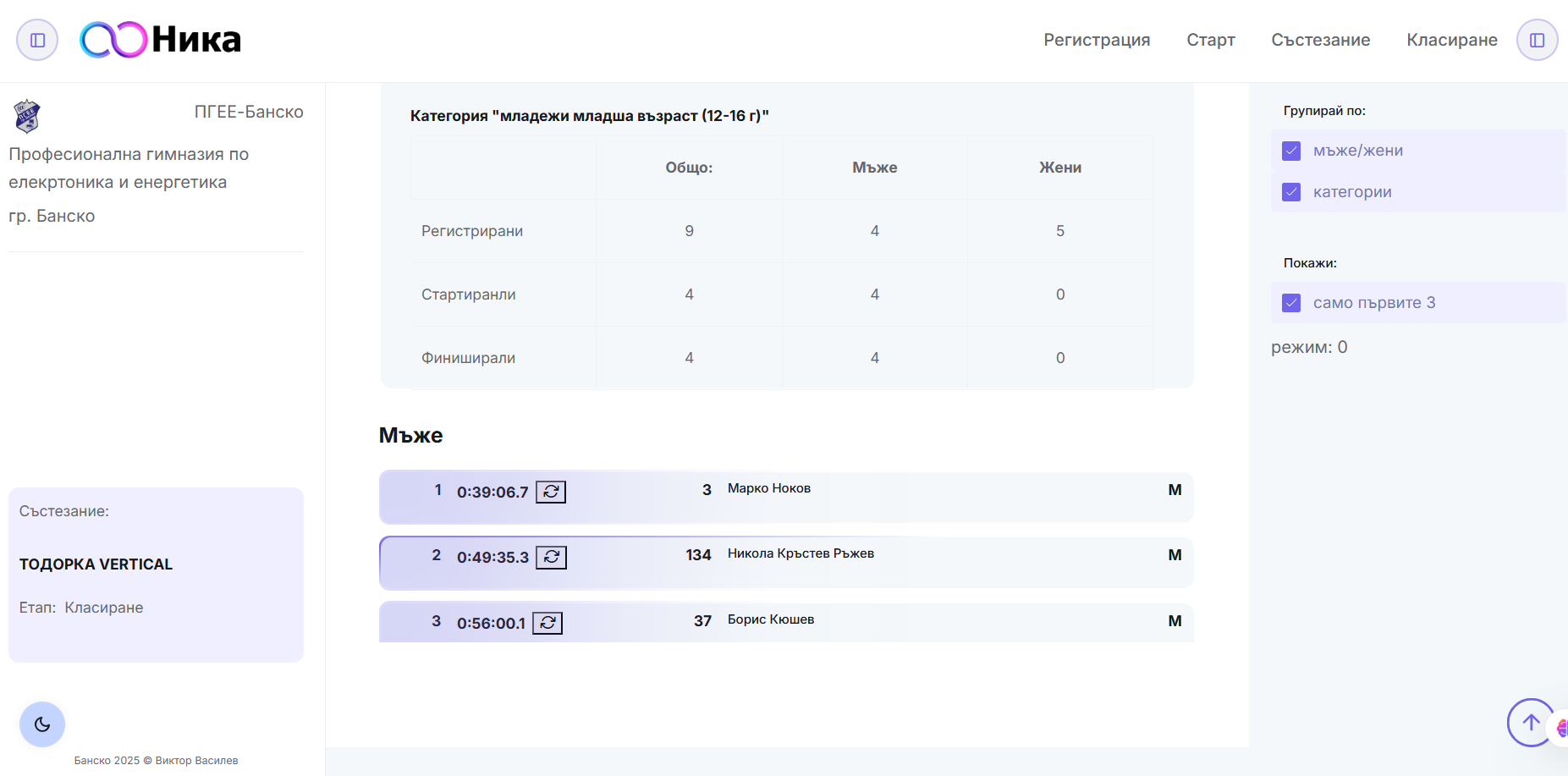
* Единен изглед, няколко режима: showMode управлява четири етапа:



0 – Регистрация 

1 - Старт



2 - Състезание 

3 - Класиране.

Компонентите се оцветяват и подреждат според текущия режим, без презареждане на страници.

* Живи данни от бекенда: периодични заявки през axios опресняват системните параметри, стартовия списък и фотофиниш кадрите. Таймерът се синхронизира със сървъра и се визуализира с точност 0.1 сек.
* Операторски действия в контекст: от един екран се добавят/редактират участници, маркира се дисвалификация, преминаване във „финиширащ“, финал, ротация на поредни номера, както и край на регистрация/състезание.
* Класиране и визуализация: налични са филтри, групиране по пол/категории, показване само на първите 3, и фотоиконка към състезател със снимки от фотофиниша.
* За всяко действие е предвиден индикатор, указващ дали действието е извършено от оператор (бадж „М“) или автоматично от робота (бадж „А“).

### Ключови елементи от имплементацията

* Инициализация и SDI логика с showMode

const App = {

delimiters: ['[[', ']]'], // избягваме конфликт с Django шаблоните

data() {

return {

sysParams: { id: 0, status: 0, next\_num: 0 },

showMode: 0, // 0 Регистрация; 1 Старт; 2 Състезание; 3 Класиране

startList: [], groupsList: [], photoList: [],

// ... други полета за филтри, таймер и модални диалози

}

},

created() {

this.loadSysParams()

this.loadGroupsList()

this.startUpdateLoop() // периодично опресняване през axios

this.updateMode = true

}

}

Vue.createApp(App).mount('#main')

* Периодично опресняване и синхронизация на часовник

loadSysParams() {

const old\_status = this.sysParams.status

axios.get('/api/sysparams/').then(resp => {

this.sysParams = resp.data[0]

if (old\_status === this.showMode) this.showMode = this.sysParams.status

if (this.sysParams.status === 2) { this.fetchStartTime(); this.fetchPhotoList() }

if (this.sysParams.status === 3 && this.timerInterval) clearInterval(this.timerInterval)

this.loadStartList()

this.fetchPhotoList()

})

},

startUpdateLoop() {

if (this.updateInterval) clearInterval(this.updateInterval)

this.lupdateInterval = setInterval(() => this.loadSysParams(), 1000)

},

fetchStartTime() {

axios.get('/api/competition/time/').then(resp => {

this.startTime = new Date(resp.data.start\_time).getTime()

this.serverNow = new Date(resp.data.server\_time).getTime()

this.timeOffset = Date.now() - this.serverNow

this.startTimerLoop()

})

},

startTimerLoop() {

if (this.timerInterval) clearInterval(this.timerInterval)

this.timerInterval = setInterval(() => {

this.timerValue = ((Date.now() - this.timeOffset) - this.startTime) / 1000

}, 100) // точност 0.1 сек

}

* Операторски промени на статуса и бързо пренареждане

setStatus(idx, value) {

this.editAthlete(idx)

if (value === 2) this.incrementNextNum() // пореден номер за „финиширащ“

this.c\_athlete.status = value

if (value === 3) { // финал

this.c\_athlete.result\_time = this.formatTimer(this.timerValue)

this.c\_athlete.num = 0

} else if (value === 0) {

this.c\_athlete.result\_time = 'DQ'; this.c\_athlete.num = 9990

} else if (value === -1) {

this.c\_athlete.result\_time = 'NS'; this.c\_athlete.num = 9999

} else {

if (value === 1) this.c\_athlete.num = 999

this.c\_athlete.result\_time = "-:--:--.-"

}

if (value === 2) this.c\_athlete.num = this.sysParams.next\_num

this.saveAthlete()

this.focusDivById('r\_' + this.c\_athlete.bib\_number)

},

swapNums(i1, i2) {

const payload = {

id1: this.startList[i1].id, num1: this.startList[i2].num,

id2: this.startList[i2].id, num2: this.startList[i1].num

}

axios.post('/api/athletes/bulk-num-update/', payload, { headers:{'X-CSRFToken': CSRF\_TOKEN} })

.then(() => { this.loadStartList() })

}

* Филтри, търсене и фотофиниш в режим „Състезание“

<div v-if="showMode == 2" class="rbt-right-side-panel">

<div class="right-side-top"><h2 class="title">[[ formatTimer(timerValue) ]]</h2></div>

<div class="chat-history-section">

<h6 class="title">Търсене:</h6>

<input id="searchBibNumber" v-model="lookupNumber"

@input="focusDivById('r\_' + lookupNumber)" class="form-control">

</div>

<div class="chat-history-section">

<h6 class="title">Покажи:</h6>

<!-- чекбокси за финиширали, по трасето, дисквалифицирани и т.н. -->

</div>

</div>

* Финален изглед и групиране

updateGroupStats() {

// определя режим на групиране според групи/пол

if (this.groupByGender) this.final\_show\_mode = this.groupByCat ? 0 : 1

else this.final\_show\_mode = this.groupByCat ? 2 : 3

// изчислява статистики за групите и присвоява състезателни места

// (setRankingNums\_GC, setRankingNums\_G, setRankingNums\_C, setRankingNums\_)

this.countFinish()

}

Защо този фронтенд е удобен

* Една страница, ясен поток: операторът не „скача“ по екрани – сменя етапи и вижда подходящите инструменти и списъци на момента.
* Реално време и прозрачност: таймерът, списъкът на финиширащите и фотофиниш изображенията се обновяват автоматично.
* Минимална зависимост: Vue + axios върху стандартен HTML шаблон; лесен за поддръжка и разширения.

С две думи, интерфейсът е фокусиран, бърз и практически ориентиран: показва живи данни, дава нужните контроли точно когато са нужни, и запазва работата на оператора в един чист, SDI-подобен екран.

## Синхронизация: веднага след събитие, навсякъде

Синхронизацията е лека и надеждна: използваме REST API и кратки периодични заявки, за да отразим промени почти мигновено на всички клиенти.

### Как работи накратко

* „Роботът“ изпраща към бекенда събитие с време, снимка и метаданни веднага щом засече финиш.
* Бекендът записва промяната и връща актуални статуси/идентификатори.
* Фронтендът извиква периодични обновявания през axios (интервал ~1 сек) и синхронизира локалното състояние: статус на състезанието, стартов списък, класиране, фотофиниш кадри.
* При преминаване в ключови режими (например старт) фронтендът прави моментна синхронизация на часовника със сървъра, за да показва коректен таймер до 0.1 сек.

### Защо е ефективно

* Ниска сложност, висока реактивност: без уебсокети и допълнителни услуги – само предвидими REST ендпойнти.
* Консистентност за всички: всеки оператор или зрителски екран вижда промяната до секунди след реалното събитие.
* Устойчиво офлайн/онлайн: при временна мрежова пауза клиентите „наваксват“ при следващата заявка.

## Заключение

НИКА започна като прагматичен отговор на един конкретен проблем — как да направим времеизмерването на състезания по-надеждно, по-достъпно и по-прозрачно — и постепенно се превърна в платформа, която свързва хардуер, софтуер и хора в едно работещо цяло. Днес имаме прототип, минал през реално събитие, с ясна архитектура и лек, управляем поток от данни: IR бариера дава точния момент, „роботът“ разпознава и маркира, бекендът пази и организира, а фронтендът показва и позволява корекции. Всичко това — офлайн, бързо и последователно.

Отвъд директната полза за организаторите, НИКА вече изпълнява и образователна мисия. Тя е флагманската ни учебна линия за 8–12 клас, която комбинира уеб програмиране, изкуствен интелект и електроника. Проектът служи като демонстрационна платформа и като практически модул за компютърно зрение в реално време, обработка на събития и интеграция на външни хардуерни решения. Учениците не просто „виждат“ технологии — те ги сглобяват, свързват и пускат в реални условия.

Гледаме напред с конкретни цели: по-устойчив OCR при трудни условия, по-богати отчети за организаторите, по-лесен импорт и автоматизация на документи, и донастройка на интерфейса за още по-бърза операторска работа. Очаквани бъдещи резултати са екипни проекти с реална приложимост и силна подготовка за национални участия. С всяка итерация НИКА става по-зряла, но запазва онова, което я прави полезна: простота, яснота и способност да работи на терен.

В крайна сметка НИКА е повече от инструмент — тя е общ език между организаторите, инженерите и учениците. Език, в който събитията се измерват точно, данните говорят ясно, а екипите растат с увереност, че правят нещо смислено и приложимо.