Online IDE

Оваа документација служи за прелгед на проектот Online IDE, детално опишувајќи ги карактеристиките, функционалностите и технологиите што се користат за развој на оваа веб апликација. Online IDE е робусна front-end веб апликација, чија што цел е да им обезбеди на корисниците целосно уредување на кодот како и средина за соработка со повеќе корисници во нивниот прелистувач. Почетната страна служи како еден вид централа за управување со проектите на корисниците, дозволувајќи им да креираат, организираат, уредуваат и бришат папки и темплејти за код. Сите овие алатки се пристапни преку модални интерфејси за полесно менаџирање. Клучна карактеристика е можноста за избор на програмски јазици за темплејтите, нудејќи флексибилност за различни потреби за програмирање. Бидејќи проектот е целосно client-side, сите податоци генерирани од корисникот, вклучувајќи папки и темплејти, постојано се чуваат во локалната меморија на прелистувачот на корисникот.

По отворањето на шаблонот, на корисниците им се претставува целосно опремен код едитор, прикажан и управуван од Monaco Editor. Овој моќен едитор нуди означување на синтаксата и прилагодливи теми (светол/темен режим) во зависност од удобноста на корисникот. Во едиторот, корисниците можат да извршуваат различни операции: да го менуваат насловот на темплејтот, да го зачувуваат својот код, доколку се премислат да го променат и програмискиот јазик, да го вклучуваат режимот на цел екран за импресивно искуство со прогрмирање, да увезуваат и извезуваат датотеки со код, да управуваат со влезот за нивните програми, да извезуваат излези од програмата и да го извршуваат својот код користејќи го Judge0 API за компилација и извршување.

Како истакната карактериситка на ова Online IDE е неговата можност за соработка на корисницте во реално време. Корисниците можат да иницираат сесија „Start New Collaboration “, која генерира единствена URL адреса за тековниот темплејт. Споделувањето на оваа URL адреса им овозможува на повеќе корисници да се приклучат на сесијата и истовремено да соработуваат на истиот код. Оваа синхронизација во реално време им овозможува на учесниците да го набљудуваат пишувањето едни со други во реално време и им дава целосни дозволи за пишување, поттикнувајќи интерактивна средина за програмирање. Сите функционалности на едиторот, како што се избор на јазик, режим на цел екран и опции за увоз/извоз, остануваат достапни за време на сесиите за соработка. Откако ќе заврши сесијата за соработка, учесниците преку копчето „Save Session As New Template” можат да ја зачуваат споделената работа како нов темплејт во нивната локална меморија на прелистувачот. Апликацијата бара име на темплејтот и им овозможува на корисниците да наведат постоечка папка или да креираат нова за зачувување. Оваа

функционалност за соработка во реално време е интегрирана со користење на Firebase Realtime Database.

Проектот е изграден врз основа на следниот технолошки стек, односно Create React App за креирање на проектот и React за конструирање на корисничкиот интерфејс. JavaScript служи како примарен програмски јазик, поткрепувајќи ја основната логика на апликацијата. MonacoEditor е искористен поради неговите напредни можности за уредување код, обезбедувајќи функции како што се означување на синтаксата и тематизирање. За компајлирање и извршување на кодот низ различни програмски јазици, проектот интегрира со Judge0 Online API.

Опис на компоненти

createCardModal.js

CreateCardModal.js претставува модален (pop-up) прозорец преку кој корисникот може да креира нов проект или темплејт. Овој модал се прикажува кога ќе се повика одредена акција во апликацијата (на пр. притискање на копче за креирање нов проект), и овозможува на корисникот да внесе име на проект и да избере програмски јазик. Компонентата CreateCardModal користи React Hooks, вклучително и useContext, useRef и useEffect. Преку ModalContext, компонентата има пристап до функцијата closeModal, со која се затвора модалот, и modalPayload, која содржи податоци поврзани со модалот. Со ProjectContext се повикува функцијата createProject, која креира нов проект со уникатен ID (генериран со помош на uuid библиотеката), наслов (title), избраниот програмски јазик и предодреден код (default code snippet) за тој јазик. Компонентата содржи и логика за автоматско поднесување на формата со копчето Enter, како и за затворање на модалот со Escape. Ова е направено преку useEffect, каде се додава и отстранува listener на настани (keydown) при прикажување на модалот. Формата во модалот содржи две главни полиња: поле за внесување на име на проектот и dropdown мени за избор на јазик (C++, Java, JavaScript или Python). При поднесување на формата, се верификува дали е внесено име, и ако е сè во ред, се креира нов објект file кој се испраќа во функцијата createProject, по што модалот се затвора. Сето ова овозможува динамично и интуитивно креирање нови проекти во Online IDE, подобрувајќи го корисничкото искуство со лесна навигација и поддршка за повеќе програмски јазици.

createFolderModal.js

Фајлот CreateFolderModal.js претставува модална форма преку која корисникот може да креира нова папка. Оваа функционалност е дел од системот за управување со фајлови во IDE-то, овозможувајќи структура слична на реална работна околина каде што фајловите може да се организираат во папки. Компонентата CreateFolderModal користит React Hooks за управување со состојби и интеракција. Се користи useContext за пристап до два контекста: ModalContext кој овозможува затворање на модалот преку функцијата closeModal и

ProjectContext кој што обезбедува пристап до функцијата createNewFolder, која креира нова папка во системот на проектот.

Покрај тоа, користејќи useRef, компонентата добива директен пристап до вредноста внесена од корисникот во полето за име на папката. Овој пристап е поефикасен за ваков тип на кратки и едноставни форми. Клучната функција onSubmitModal се повикува кога корисникот ја поднесува формата. Таа спречува стандардно освежување на страната, ја чита и валидаира внесената вредност (се проверува дали е празна), и ако се е во ред, ја повикува createNewFolder функцијата со името на папката, а потоа го затвора модалот.

Во useEffect hook се поставува listener на настани од тастатура, кој овозможува затворање на модалот со копчето Escape и автоматско поднесување на формата при притискање на Enter, ако е внесено валидно име на папката.

Визуелниот дел на компонентата е HTML-форма со CSS класи, при што корисникот има јасна и едноставна можност да внесе име на папка и да ја креира. Копчето "Create" е со стил и поставено веднаш под полето за внесување, а иконата "X" овозможува рачно затворање на модалот.

createProjectModal.js

Компонентата CreateProjectModal.js служи за креирање на нов проект преку модален прозорец кој се појавува кога корисникот ќе избере опција за додавање нов проект. Компонентата користи контексти за управување со модалниот прозорец и за креирање проекти. Преку ModalContext, корисникот може да го затвори модалот, додека ProjectContext ја обезбедува функцијата createNewProject која се повикува кога корисникот ќе ја поднесе формата. Формата што се наоѓа во модалот содржи три полиња за внесување: првото е за името на фолдерот во кој ќе се смести проектот, второто е за името на темплејтот, а третото е селектор за избор на програмски јазик – C++, Java, JavaScript или Python. Овие вредности се собираат преку референци (useRef), и потоа се користат при поднесување на формата. Доколку некоја од вредностите не е внесена, функцијата нема да продолжи со креирање на проектот. Кога ќе се притисне копчето Create или кога корисникот ќе притисне Enter, се активира функцијата onSubmitModal која ги чита внесените податоци, ги проверува и ако се валидни, повикува креирање на нов проект преку createNewProject, по што модалот се затвора. Дополнително, во компонентата е поставен listener на тастатура со useEffect hook, кој овозможува користење на Escape за затворање на модалот и Enter за автоматско поднесување на формата, што го прави користењето на модалот побрзо и поудобно. Оваа компонента игра важна улога во апликацијата, бидејќи преку неа корисниците лесно и ефикасно можат да започнат нов проект во рамки на IDE-то, избирајќи го јазикот и организациската структура по своја желба.

Modal.js

Компонентата Modal.js ја содржи една од поважните улоги во управувањето и прикажувањето на модалните прозорци. Таа ги координира сите можни модали кои можат да се појават во апликацијата, како што се модалот за креирање на проект, фолдер, ажурирање на наслов на фолдер или фајл, и креирање на нова картичка (код фајл). Главната задача на оваа компонента е да провери кој тип на модал е активен во моментот и да го прикаже соодветниот модален прозорец на екранот. Компонентата користи useContext за да добие пристап до ModalContext, кој ги содржи сите информации за тековната состојба на модалниот систем, вклучувајќи го и activeModal. Потоа, преку условни рендерирања, проверува дали activeModal се совпаѓа со некој од дефинираните типови од modalConstants, како што се CREATE_PROJECT, CREATE_FOLDER, UPDATE_FOLDER_TITLE, UPDATE_FILE_TITLE, и CREATE_CARD. Ако се совпаѓа, тогаш се прикажува соодветната модал компонента, на пример CreateProjectModal или UpdateFileTitleModal. Овој пристап овозможува сите модали да се управуваат од едно централно место, без да се распрснува логиката низ различни делови од апликацијата. Ова го прави кодот поорганизиран, полесен за одржување и проширување. Доколку во иднина се додаде нов тип на модал, доволно е само да се импортира и да се додаде нов услов во Modal компонентата. На овој начин, Modal.js служи како модуларен контролер за UI дијалозите, обезбедувајќи динамичен и флексибилен начин за интеракција на корисникот со различни функционалности во IDE-то.

updateFileTitleModal.js

UpdateFileTitleModal компонентата е наменет модален интерфејс кој им овозможува на корисниците да преименуваат постоечки темплејти на код. Користи React Context API за да комуницира и со ModalContext (за контрола на видливоста на модалот) и со ProjectContext (за извршување на промената на насловот на фајлот).

При рендерирање, модалот прикажува поле за внес каде што корисникот може да го впише новиот наслов. Се користи inputRef за директен пристап до вредноста од полето. Кога ќе се поднесе формата (со клик на "Save" или притискање Enter), се повикува функцијата onSubmitModal. Оваа функција го спречува стандардното однесување на формата, ја зема вредноста од внесот (trim-ирана), и ако не е празна, ја повикува функцијата editFileTitle од ProjectContext со новиот наслов, заедно со folderId и fileId добиени од modalPayload. По ажурирањето, модалот автоматски се затвора преку closeModal.

useEffect hook додава глобални listeners за тастатура: притискањето Escape го затвора модалот, а Enter се обидува да ја поднесе формата.

updateFolderTitleModal.js

UpdateFolderTitleModal компонентата претставува модален интерфејс прилагоден за корисници, наменет за преименување на постоечки папки од проекти во рамки на Online IDE. Компонентата го управува своето прикажување и операциите за ажурирање на папките со користење на React Context API, пристапувајќи до ModalContext за контрола на модалот и до ProjectContext за манипулација со податоците од проектот.

Модалот содржи поле за внес, контролирано преку inputRef, каде што корисниците го внесуваат новиот наслов на папката. При поднесување на формата било со клик на копчето „Save“или со притискање на Enter се повикува функцијата onSubmitModal. Оваа функција најпрво го спречува стандардното однесување на формата, потоа ја зема и прочистува вредноста од внесот. Доколку новиот наслов не е празен, се повикува акцијата editFolderTitle од ProjectContext, со новиот наслов и modalPayload (кој го содржи уникатниот идентификатор на папката) за да се ажурираат податоците на проектот. По успешната промена, се повикува closeModal за да се затвори модалот.

Дополнително, useEffect hook имплементира глобални кратенки од тастатурата за подобрена употребливост: копчето Escape овозможува брзо затворање на модалот, додека Enter го активира поднесувањето на формата ако полето за внес не е празно.

ModalProvider.js

Фајлот кој го дефинира ModalContext и ModalProvider игра улога во управувањето со модалните прозорци низ целата апликација. Оваа логика е имплементирана преку React Context API со цел да се овозможи управување со модалите, без потреба модалите да се контролираат од секоја компонента поединечно. Во ModalContext се создава глобален контекст со помош на createContext, што овозможува секоја компонента во апликацијата да има пристап до состојбата поврзана со модалите. Потоа се дефинираат modalConstants, кои претставуваат различни типови на модали што може да се прикажат, како што се креирање проект, фолдер, ажурирање на наслов и други. Овие константи се користат за да се знае кој конкретен модал треба да биде прикажан во даден момент. Компонентата ModalProvider ги чува двата главни податочни елементи: modalType, кој го претставува активниот тип на модал, и modalPayload, кој претставува дополнителни податоци поврзани со модалот (на пример, ID на фолдер или фајл што се уредува). Со useState се управува со овие вредности и се дефинира функцијата closeModal, која ги ресетира на null, што резултира со затворање на модалот. Контекстот потоа ги изложува сите функционалности преку modalFeatures, како што се openModal, closeModal, activeModal и setModalPayload. Секој од овие методи може да се користи во другите компоненти преку useContext(ModalContext), со што се добива лесна контрола врз отворањето и затворањето на модални прозорци. ModalProvider го обвиткува целото дете-содржина со ModalContext.Provider, што значи дека сите компоненти во апликацијата кои се внатре во овој провајдер ќе имаат пристап до контекстот и можност за интеракција со модалниот систем. Со оваа архитектура, апликацијата добива

флексибилност, одржливост и доследност во приказот на модали, без непотребно повторување на логика.

ProjectProvider.js

ProjectProvider компонентата управува со сите податоци и функционалности поврзани со проектите. Ја користи React Context API за глобално споделување на проектните папки, фајлови и различни функции за нивна манипулација, достапни за секоја компонента во нејзиниот опсег. Овој провајдер обезбедува континуитет на податоците преку корисничките сесии со користење на localStorage.

При иницијализација, ProjectProvider се обидува да ги вчита податоците за проектот од localStorage. Доколку не се пронајдат податоци, се користи однапред дефинирана структура initialData, која вклучува пример-папки и шаблони на код. Дополнително, се дефинирани и defaultCodes за повеќе програмски јазици, кои овозможуваат иницијални "Hello World" примери при креирање на нов фајл.

ProjectProvider опфаќа богат сет на функции за управување со структурата на проектот:

- createNewProject: Додава нов шаблон на код. Проверува дали папката веќе постои – ако постои, го додава шаблонот таму; ако не, креира нова папка.
- createNewFolder: Додава нова, празна папка во проектот.
- deleteFolder: Брише цела папка заедно со сите нејзини фајлови.
- editFolderTitle: Ја ажурира ознаката на постоечка папка.
- editFileTitle: Го преименува конкретен шаблон на код во рамки на папка.
- deleteFile: Брише поединечен шаблон од одредена папка.
- createProject: Општа функција за додавање фајл во постоечка папка.
- getDefaultCode, getFileName, getLanguage: Утилити функции за добивање на кодот, името или програмскиот јазик на фајл според неговото ID и ID на папката.
- updateLanguage: Ја менува програмската јазична поставка на шаблонот и го ажурира кодот со соодветен "Hello World" пример.
- saveCode: Го зачувува тековниот код на даден шаблон.

Сите промени направени преку овие функции веднаш се синхронизираат со state-от на компонентата (folders) и се перзистираат во localStorage користејќи JSON.stringify. Дополнително, useEffect hook се грижи иницијалните податоци да се зачуваат во localStorage доколку претходно не постојат. Преку обезбедување на projectFeatures преку ProjectContext.Provider, ProjectProvider овозможува робустен и централизаран механизам за управување со целиот работен простор на проекти во рамки на Online IDE.

editor.js

Компонентата EditorTemplate претставува една од најважните компоненти во проектот, имено дизајнирана е да обезбеди околина за уредување на кодот, негово извршување и колаборација во реално време. Оваа компонента управува со низа внатрешни состојби кои се клучни за функционирањето на апликацијата, вклучително и содржината на полињата за кориснички влез (input) и излез од програмата (output), активирање и деактивирање на loading indicator за време на фазите на компалирање и извршување на кодот, како и со прецизно справување со грешки кои може да настанат при поднесување на кодот. Дополнително, во компонентата се интегрира и со Firebase, што ги овозможува уникатните функции за реална колаборација на апликацијата. Во основата, компонентата користи React useState hook за влез и излез, директно управувајќи со содржината прикажана во соодветните текстуални полиња. Boolean вредноста showLoader е фактор за обезбедување на моментален визуелен фидбек до корисникот, прикажувајќи full-page loading spinner секогаш кога се извршуваат процеси што бараат поголема компјутерска моќ, како компилација на код преку Judge0 API. Исто така, Boolean вредноста isError, кога е поставена на true, означува дека последното извршување на кодот резултирало со грешка, што предизвикува визуелна промена во излезниот приказ, обично прикажувајќи пораки за грешка со црвена боја за подобра прегледност и внимание на корисникот.

Функцијата importInput овозможува лесно вчитување на надворешни текстуални фајлови од страна на корисникот, парсирајќи ја нивната содржина и динамички ажурирајќи ја состојбата на влезот. Оваа функција е особено корисна за тестирање код со специфични datasets или однапред дефинирани сценарија. Од друга страна, функцијата exportOutput му овозможува на корисникот да го преземе резултатот од извршувањето на програмата како plain text file, именуван како "output.txt". Проверка за валидност внимателно се прави пред преземање за да се избегне креирање на празни датотеки. useCallback hook е клучен дел од процесот на извршување, прецизно ракувајќи со асинхроните одговори од сервисот за поднесување на код. За време на компајлирање, showLoader се поставува на true, а по прием на одговор од Judge0 API, функцијата интелигентно го декодира и ажурира output со stdout или stderr, соодветно прилагодувајќи го isError. Тука е неопходна и atob() функцијата, бидејќи Judge0 ги енкодира своите одговори во Base64. Врз ова се надоврзува и runCode hook, кој служи како главен тригер за извршување на кодот – тој ги спакува моменталниот код, селектираниот програмски јазик, и евентуалниот влез, и го испраќа до makeSubmission сервисот за обработка. Callback функцијата потоа го обработува добиениот одговор, затворајќи го циклусот на извршување.

Можеби најиновативниот аспект на EditorTemplate е функцијата за real-time collaboration, иницирана преку startNewCollaborationSession. Оваа функција започнува live coding session со генерирање на уникатен session ID преку uuidv4. Потоа креира нов запис во Firebase Realtime Database под патеката sessions/{newSessionId}, иницијализирајќи го со моменталната содржина на кодот и избраниот јазик. По успешната поставка на Firebase,

прелистувачот на корисникот се пренасочува кон нов URL кој го содржи session ID како query parameter. Веднаш потоа, корисникот добива известување со споделив collaboration URL, што му овозможува да покани други учесници во сесијата. Целата оваа механика се потпира на моќните капацитети на Firebase за синхронизација на податоци во реално време, осигурувајќи дека сите учесници ја гледаат и можат интерактивно да ја уредуваат истата верзија на кодот.

Логиката за прикажување на компонентата е елегантно структурирана: горниот дел содржи хедер со логото на апликацијата, централниот дел е посветен на EditorContainer – кој го содржи Monaco Editor. Од страните се поставени оддели за input и output, секој со јасни хедери, визуелно привлечни import/export buttons (со FontAwesome икони) и текстуални полиња со прилагодлив одзив. Условното прикажување на full-page loader обезбедува фино корисничко искуство при позадински операции. EditorContainer прима сет од props, вклучително и file и folder IDs добиени од URL, runCode функцијата, sessionId за колаборација, како и посебни handlers за започнување нови сесии или проекти.

EditorContainer.js

Компонентата EditorContainer е клучен дел од Online IDE, која го обвива Monaco Editor и овозможува богат сет на функционалности за индивидуално и колаборативно уредување и управување со кодот. Оваа компонента служи како директен родител на самиот едитор, управувајќи со јазикот, темата и содржината, додека истовремено се справува и со интеракциите поврзани со зачувување, вчитување, преземање и извршување на код. Интелегентно го прилагодува своето однесување во зависност од тоа дали корисникот е во стандарден режим за уредување или во real-time collaboration сесија, користејќи React state и effect hooks во комбинација со Firebase Realtime Database за синхронизација на податоци.

При иницијализација, EditorContainer ги превзема тековниот јазик и содржината на темплејтот преку context providers, осигурувајќи дека едиторот се вчитува со точната почетна состојба. Клучните состојби со кои управува компонентата се: language и code за содржината и syntax highlighting на уредувачот, theme за светол или темен режим, isFullScreen за проширување на уредувачот, и isEditingTitle за директно преименување на темплејтот. Се користи и useRef hook (codeRef) за да се одржи променлива референца до моменталниот код – особено важно за избегнување непотребни повторни рендерирања и за ефикасно ажурирање на Firebase при колаборација. useEffect hook-овите се стратешки поставени: еден се грижи јазикот и кодот да се синхронизираат при промена на fileId или folderId, овозможувајќи фини транзиции помеѓу различни темплејти. Друг клучен useEffect hook го менаџира listener-от на Firebase Realtime Database; кога има sessionId, се претплатува на промени во кодот и јазикот на споделената сесија. Оваа реална синхронизација е основа за заедничко уредување, динамички ажурирајќи го локалниот едитор кога другите учесници прават промени. Дополнителни useEffect hook-ови управуваат со распоредот и адаптивноста

на уредувачот, особено при премин во и од fullscreen, осигурувајќи дека Monaco Editor правилно се ресајзира.

onChangeCode ја ажурира локалната состојба на кодот и codeRef, а доколку има активна колаборациска сесија, овие промени се испраќаат и до Firebase, обезбедувајќи ги real-time updates за сите поврзани корисници. Функцијата importCode овозможува вчитување на код од локални датотеки во уредувачот. Таа содржи валидација за дозволени програмски формати (на пр. .cpp, .js), и ако сесијата е колаборативна, вчитаниот код се испраќа и во Firebase. Обратно, exportCode овозможува преземање на моменталниот код како датотека, автоматски додавајќи соодветен екстензија според селектираниот јазик. onChangeLanguage и onChangeTheme им овозможуваат на корисниците да го променат програмскиот јазик за syntax highlighting и да изберат светла или темна тема – промената на јазикот исто така се синхронизира преку Firebase при колаборативни сесии. onSaveCode го зачувува моменталниот код во локалното складиште на прелистувачот. Функцијата handleEditorDidMount овозможува пристап до Monaco Editor instance, што е важно за програмска контрола, како на пр. прилагодување на распоредот. toggleFullScreen овозможува проширување на едиторот на цел екран, со handleEscKey listener за брзо излегување со Escape копчето од тастатура. onRunCode го иницира надворешниот процес за извршување на кодот преку проп функција runCode, испраќајќи го моменталниот код и избраниот јазик.

За активности поврзани со колаборација, onStartNewCollaborationClick започнува нова споделена сесија со повикување на проп функција, испраќајќи го тековниот код и јазик како иницијални вредности. saveSessionAsNewTemplate нуди начин корисниците во колаборативна сесија да го зачуваат заедничкиот код како нов независен template во нивниот локален browser storage, со барање за внес на име на темплејтот и селекција на фолдер (со "Saved Sessions" како стандардна опција). Ова овозможува вредната колаборативна работа лесно да се зачува и повторно да се користи, дури и ако корисникот нема постоечка проектна структура за сесијата.

service.js

Овој фајл во рамките на Online IDE е задолжен за целата комуникација со Judge0 API — надворешен сервис кој се справува со компилација и извршување на код. Неговата примарна улога е да ја сосредочи логиката за поднесување на корисничкиот код и добивање на резултати, со што ги апстрахира сложеностите поврзани со интеракцијата со надворешниот API. Ова овозможува почист и полесно одржлив код за остатокот од апликацијата.

Клучен елемент во овој модул е languageCodeMap — објект кој ефикасно ги преведува вообичаените имиња на програмски јазици (на пр. 'cpp', 'javascript') во специфични

нумерички ID вредности кои ги бара Judge0 API. Ова мапирање е клучно за точно идентификување на програмскиот јазик на поднесениот код.

Модулот нуди две главни асинхрони функции, односно getSubmission(tokenId, callback) – Оваа функција е наменета за polling на Judge0 API. Откако ќе се поднесе код и се добие tokenId, оваа функција повеќекратно го повикува Judge0 додека статусот на извршување не биде финален, враќајќи ги деталите за извршувањето или грешка преку дадениот callback. makeSubmission({code, language, callback, stdin}) – Оваа функција иницира нова компилација на код. Прво врши валидација дали постои код, потоа креира и испраќа POST барање до Judge0 API, каде source code-от и standard input (stdin) се енкодирани во Base64. По добивањето на tokenId, makeSubmission влегува во polling циклус, повторно повикувајќи getSubmission со стратегиски паузи за да избегне прекумерни API барања. Откако ќе се добие финалниот резултат, функцијата ги обработува сите compile_output или stderr пораки, декодирајќи ги од Base64. На крај, го повикува callback со apiStatus (на пр. 'loading', 'success', 'error') и соодветни податоци или порака за грешка, овозможувајќи на UI-то да го прикаже резултатот од извршувањето.

RightComponent(index.js)

Овој фајл дефинира две React компоненти, односно Folder и RightComponent кои заедно ја сочинуваат секцијата „My Projects“ во корисничкиот интерфејс на апликацијата. Тие се одговорни за прикажување и управување со кориснички креирани папки и шаблони на код. Двете компоненти интензивно ја користат React Context API за поврзување со ProjectContext (за манипулација со проектните податоци) и ModalContext (за контрола на различни модални дијалози).

Компонентата Folder е функционална компонента што прима folderTitle, cards (низа од фајл-објекти), и folderId како пропс. Таа овозможува функционалности за управување со поединечни папки и нивната содржина. Преку useContext пристапува до функциите deleteFolder и deleteFile од ProjectContext за бришење на папки и фајлови. Ја користи openModal и setModalPayload од ModalContext за активирање на различни модални дијалози, како што се: UPDATE_FOLDER_TITLE за преименување на папка и CREATE_CARD за додавање нов шаблон во папката.

Функциите onDeleteFolder и onEditFolderTitle се користат за бришење и уредување на папката. За секоја „картичка“ (фајл) во папката, рендерира кликабилен елемент. Клик на картичката пренасочува кон страницата за уредување на кодот со useNavigate, испраќајќи file.id и folderId како параметри во URL-то. Секоја картичка вклучува копчиња за onDeleteFile и onEditFile, со логика што спречува пренесување на кликот кон родителот (за да не се активира ненамерна навигација). Визуелната структура вклучува икона на папка, наслов, и акциски копчиња за бришење, уредување и додавање нов шаблон.

Компонентата RightComponent е компонента која ја организира прикажаната структура на сите проектни папки. Преку ProjectContext пристапува до низата folders, а преку ModalContext до функциите за контрола на модални прозорци. Прикажува истакнат наслов „My Projects“ и копче „New Folder“, кое го отвора CREATE_FOLDER модалот преку openModal. Потоа ги итерира сите објекти во folders, динамички рендерирајќи Folder компонента за секоја поединечна папка. Пропс-ите folderTitle, cards и folderId се проследуваат до секоја инстанца на Folder, со што се овозможува организирано прикажување и управување со сите проекти на корисникот.

HomeTemplate(index.js)

Компонентата HomeTemplate ја претставува главната структура на почетната страница во Online IDE, обезбедувајќи основен распоред и клучни интерактивни елементи за управување со корисничките проекти за програмирање.

Распоредот е поделен на два главни дела Лев контејнер: Прикажува лого на апликацијата. Содржи копче „Креирај нов проект“ кое при клик го повикува openCreateProjectModal. openCreateProjectModal ја користи функцијата openModal од ModalContext (преку useContext) за прикажување на модал прозорец специјално дизајниран за создавање нов проект, идентификуван со modalConstants.CREATE_PROJECT.

Десен контејнер: Се рендерира преку компонентата RightComponent, која е увезена и вметната во HomeTemplate. Како што претходно беше опишано, RightComponent е одговорна за прикажување и управување со корисничките проектни папки и шаблони на код.

На крајот од компонентата, се рендерира и глобалната Modal компонента (исто така поврзана преку ModalContext). Таа служи како универзален слој за прикажување на сите модални дијалози низ апликацијата, гарантирајќи правилно позиционирање и преклопување над главната содржина.

firebase.js

firebase.js претставува конфигурациски фајл кој служи како порта преку која Online IDE се поврзува со Google Firebase сервисите. Неговата примарна одговорност е иницијализација на Firebase апликацијата со потребните креденцијали, правејќи ги функционалностите на Firebase — особено Realtime Database — достапни низ целиот проект. Овој фајл ја импортира функцијата initializeApp од firebase/app SDK. Потоа се дефинира објектот firebaseConfig, кој ги содржи сите специфични конфигурациски детали потребни апликацијата да се автентикира и комуницира со соодветниот Firebase проект. Овие детали вклучуваат: apiKey, authDomain, databaseURL (кој покажува кон Realtime Database

инстанцата во Europe West 1, клучен за функцијата за реална колаборација), projectId, storageBucket, messagingSenderId, appId.

Овие параметри заедно обезбедуваат правилно идентификување и поврзување на апликацијата со нејзиниот Firebase backend.

На крај, се повикува initializeApp(firebaseConfig) за да се иницијализира Firebase апликацијата со дадената конфигурација. Добиениот app instance потоа се експортира како default export, овозможувајќи другите компоненти — како што е EditorContainer — да го импортираат и користат Firebase (посебно getDatabase) за синхронизација на податоци во реално време при колаборативно уредување. Оваа модуларна поставеност ја централизира иницијализацијата на Firebase, промовирајќи чист код и лесно управување со конфигурациите на сервисите.

Заклучок

Како заклучок, овој Online IDE React нуди feature-rich, user-friendly и колаборативни можности за програмирање. Специфичниот интерфејс на самата апликација, сеопфатните функционалности на едиторот и можностите за соработка во реално време го прават посебен од останатите online едитори како и една поинкава алатка за индивидуални програмери и тимови, а воедно одржувајќи ја перзистентноста на податоците во прелистувачот на корисникот.