

СОДЕРЖАНИЕ

Обозначения и сокращения.....	6
Введение.....	7
1 Анализ предметной области.....	9
1.1 Основные этапы создания оригинального ЛС.....	9
1.2 Процесс разработки нового фармакологически активного вещества.....	12
1.3 Высокопроизводительный скрининг	16
1.4 Молекулярный докинг	19
2 Анализ прототипов и постановка задачи.....	23
2.1 Задачи на проектирование	23
2.2 Использование системы.....	23
2.3 Обобщённый алгоритм работы программы.....	26
3 Технологии и паттерны, используемые при разработке	27
3.1 Python.....	27
3.2 Django	31
3.3 HTML.....	33
3.4 Javascript	37
3.5 Ajax.....	43
3.6 SMILES	46
3.7 Model-View-Controller	49
4 Описание алгоритмов, реализующих бизнес-логику	52
4.2 CountVectorizer.....	52
4.3 Дерево решений	54
5 Техничко-экономическое обоснование дипломного проекта	56
5.1 Расчет сметы затрат и цены программного продукта.....	56
5.2 Расчет сметы затрат и цены программного продукта.....	56
5.3 Расчёт капитальных затрат.....	66
5.4 Расчет экономического эффекта.....	67
5.5 Вывод по технико-экономическому обоснованию.....	68
Выводы и заключения.....	69
Список использованной литературы.....	70
ПРИЛОЖЕНИЕ А (обязательное) Код классов и алгоритмов системы	71

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей пояснительной записке применяются следующие определения и сокращения.

Биомишень – это химическая структура, участвующая в биологической системе человека, работа которой нарушается при болезни организма.

Лиганд – атом, ион или молекула, связанные с неким центром (акцептором).

Ингибитор – общее название веществ, подавляющих или задерживающих течение физиологических и физико-химических (главным образом ферментативных) процессов.

Докинг – это метод молекулярного моделирования, который позволяет предсказать наиболее выгодную для образования устойчивого комплекса ориентацию и положение одной молекулы по отношению к другой.

ЛС – лекарственное средство.

ИТ – информационные технологии.

Фреймворк – программное обеспечение, облегчающее разработку и объединение различных компонентов большого программного проекта.

Динамическая типизация – приём, широко используемый в языках программирования и языках спецификации, при котором переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной.

ПО – программное обеспечение.

ООП – объектно-ориентированное программирование.

Web-сервер – сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-поток или другими данными.

WWW – World Wide Web – распределённая система, предоставляющая доступ к связанным между собой документам, расположенным на различных компьютерах, подключённых к сети Интернет. Для обозначения Всемирной паутины также используют слово веб (англ. web «паутина») и аббревиатуру WWW.

HTML – HyperText Markup Language – «язык гипертекстовой разметки».

AJAX – Asynchronous Javascript and XML – «асинхронный JavaScript и XML».

HTTP – HyperText Transfer Protocol – «протокол передачи гипертекста».

IP-адрес – уникальный сетевой адрес узла в компьютерной сети, построенной на основе стека протоколов TCP/IP.

XML – eXtensible Markup Language – расширяемый язык разметки.

MVC – схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.

ВВЕДЕНИЕ

С развитием информационных технологий и телекоммуникаций жизнь становится все более мобильной и информативной, новые технологии прочно входят в различные отрасли хозяйствования, сферы жизни и несут новые нормы в них. В связи с реформированием экономики, с взятием курса на её инновационное развитие, всё чаще и чаще в повседневной работе в большинстве отраслей начинают использовать различные средства информационно-вычислительной техники и соответственно программного обеспечения. Уровень развития медицинской отрасли в государстве – один из важнейших признаков ее технологического прогресса и цивилизованности.

Самые широко применяемые средства ИТ в наше время – это интернет, мобильные телефоны и компьютеры. Тем не менее, каждая узкая отрасль науки и производства имеет своё специально разработанное программное обеспечение, обеспечивающее работу систем и обрабатывающее информацию. Использование информационных технологий в медицине является логичным шагом, так как быстрый доступ к информации и обмен ею существенно сокращает временные издержки на поиск решений, а время часто является главным фактором в спасении жизни человека. Кроме того, применение технологий машинного обучения и нейросетей выводит её на качественно новую ступень.

Компьютер все больше используется в области здравоохранения, что бывает очень удобным, а порой просто необходимым. Благодаря этому медицина приобретает сегодня совершенно новые черты. Во многих медицинских исследованиях просто невозможно обойтись без компьютера и специального программного обеспечения к нему. Этот процесс сопровождается существенными изменениями в медицинской теории и практике, связанными с внесением корректив к подготовке медицинских работников. Современный период развития общества характеризуется сильным влиянием на него компьютерных технологий, которые проникают во все сферы человеческой деятельности, обеспечивают распространение информационных потоков в обществе, образуя глобальное информационное пространство. Они очень быстро превратились в жизненно важный стимул развития не только мировой экономики, но и других сфер человеческой деятельности. Трудно найти сферу, в которой сейчас не используются информационные технологии.

Жизненный путь каждого человека в той или иной степени пересекается с врачами, которым мы доверяем свое здоровье и жизнь. Но образ медицинского работника и медицины в целом в последнее время претерпевает сильные изменения, и происходит это во многом благодаря развитию информационных технологий.

В современных условиях при синтезе лекарственных средств возникают две фундаментальные проблемы: структурные манипуляции–синтез новых структур и способность переходить от одной структуры к другой и вторая –

соотношение структуры и свойств вещества. А так как отрасль медицины интересуется именно свойствами веществ, органическая химия призвана синтезировать всё новые и новые вещества. Именно поэтому важная задача для органической химии – синтез не веществ, а свойств. С другой стороны, основная задача медицинской химии – создание соединений с заранее заданной физиологической активностью, так называемый рациональное проектирование лекарственных средств. В этом призвано помочь машинное обучение, которое сможет найти закономерности в химических формулах веществ, с одной стороны, и качественного эффекта на организм человека при лечении заболеваний, с другой. В общем случае система позволит провести качественную оптимизацию комбинаторной библиотеки, а также упрощение молекулярного докинга.

1 СОВРЕМЕННАЯ МЕТОДОЛОГИЯ СОЗДАНИЯ ОРИГИНАЛЬНЫХ ЛЕКАРСТВЕННЫХ СРЕДСТВ

1.1 Основные этапы создания оригинального лекарственного средства

Оригинальное ЛС – лекарственное средство, которое отличается от всех ранее зарегистрированных лекарственных средств фармакологически активным веществом или комбинацией таких веществ.

Разработка оригинального лекарственного средства состоит из четырёх этапов:

- 1) разработка концепции исследования и получение активных веществ;
- 2) доклинические исследования;
- 3) клинические исследования:
 - а) фаза I;
 - б) фаза II;
 - в) фаза III;
- 4) регистрация, начало производства, запуск на рынок и реализация – фаза IV.

В таблице 1.1 приведены общие затраты на создание нового активного вещества.

Таблица 1.1 – Общие затраты на создание нового активного вещества

Этап	Стоимость	Временные затраты
Разработка, ДКИ	121 млн \$	3-7 лет
КИ I	15,2 млн \$	2-7 лет
КИ II	23,5 млн \$	
КИ III	86,3 млн \$	
ПРИ	5,2 млн \$	6-18 месяцев
ВСЕГО	251,2 млн \$	5,5–17 лет

Доклинические исследования включают в себя исследования *in vitro* (лабораторные исследования в пробирках) и исследования *in vivo* (исследования на лабораторных животных), в ходе которых исследуются различные дозы тестируемого вещества, чтобы получить предварительные данные о фармакологических свойствах, токсичности, фармакокинетике и метаболизме изучаемого препарата. Доклинические исследования помогают фармацевтическим компаниям понять, стоит ли исследовать вещество дальше. Исследования с участием людей можно начинать, если данные, полученные в

ходе доклинических исследований, доказывают, что препарат может применяться для лечения заболевания, если препарат достаточно безопасен и исследования не подвергают людей ненужному риску.

Процесс разработки лекарственного препарата часто описывается как последовательное проведение четырёх фаз клинических исследований. Каждая фаза – это отдельное клиническое исследование, для регистрации препарата может потребоваться несколько исследований в рамках одной и той же фазы. Если препарат успешно проходит испытания в первых трёх фазах, он получает регистрационное удостоверение. Исследования IV фазы – это пострегистрационные исследования.

Фаза I клинического испытания. Исследуется переносимость однократной дозы, фармакокинетика, фармакодинамика и другие свойства, не имеющие отношения к специфическому фармакологическому действию.

Исследование проходит на группе здоровых добровольцев. Проводится поиск побочных эффектов. В ходе фазы I исследуются такие показатели, как абсорбция, распределение, метаболизм, экскреция, а также предпочтительная форма применения и безопасный уровень дозирования. Фаза I обычно длится от нескольких недель до 1 года.

Есть различные типы исследований фазы I:

1) исследования однократных нарастающих доз – это исследования, в которых небольшому числу пациентов вводится одна-единственная доза исследуемого препарата в течение всего того времени, что они находятся под наблюдением. Если не обнаруживается никаких нежелательных реакций и фармакокинетические данные соответствуют ожидаемому уровню безопасности, то доза увеличивается и эту увеличенную дозу получает следующая группа участников. Введение препарата с эскалацией доз продолжается до тех пор, пока не будут достигнуты заранее намеченные уровни фармакокинетической безопасности либо не обнаружатся недопустимые нежелательные реакции (при этом говорят, что достигнута максимально допустимая доза);

2) исследования многократных нарастающих доз – это исследования, которые проводятся, чтобы лучше понять фармакокинетику и фармакодинамику препарата при многократном введении. В ходе таких исследований группа пациентов получает низкие дозы препарата многократно. После каждого введения производится забор крови и других физиологических жидкостей, чтобы оценить, как лекарство ведёт себя в человеческом организме. Доза постепенно повышается в следующих группах добровольцев – до заранее определённого уровня. Фаза II клинического испытания: исследуется оценка пользы от испытуемого вещества при данном

заболевании, определяется уровень терапевтической дозы, схемы дозирования и краткосрочной безопасности. Оценив фармакокинетику и фармакодинамику, а также предварительную безопасность исследуемого препарата в ходе исследований фазы I, компания-спонсор инициирует исследования фазы II на большей популяции. Исследование проводится на группе пациентов (40-80 человек) с заболеванием, для которых предназначено данное лекарственное средство. Дизайн исследований фазы II может быть различным, включая контролируемые исследования и исследования с контролем исходного состояния. Последующие исследования обычно проводятся как рандомизированные контролируемые, чтобы оценить безопасность и эффективность препарата по определённому показанию.

Исследования *фазы II* обычно проводятся на небольшой гомогенной (однородной) популяции пациентов, отобранной по жёстким критериям. Важная цель этих исследований – определить уровень дозирования и схему приёма препарата для исследований фазы III. Дозы препарата, которые получают пациенты в исследованиях фазы II, обычно (хотя и не всегда) ниже, чем самые высокие дозы, которые вводились участникам в ходе фазы I. Дополнительной задачей в ходе исследований фазы II является оценка возможных конечных точек, терапевтической схемы приёма (включая сопутствующие препараты) и определение таргетной группы (например, лёгкая форма против тяжёлой) для дальнейших исследований в ходе фазы II или III.

Фаза III клинического испытания. Исследуется эффективность и безопасность лекарственного средства в условиях приближенных к тем, в которых оно будет использоваться, если будет разрешено к медицинскому применению. Часть пациентов получает стандартную терапию (или плацебо при исследовании новых классов лекарственного средства) а часть – экспериментальную. Исследование проводится на большой группе пациентов (около 1-3 тыс) разного возраста пола и разной сопутствующей патологией. Эти исследования спланированы таким образом, чтобы подтвердить предварительно оценённые в ходе фазы II безопасность и эффективность препарата для определённого показания в определённой популяции. В исследованиях фазы III также может изучаться зависимость эффекта от дозы препарата или препарат при применении у более широкой популяции, у пациентов с заболеваниями разной степени тяжести или в комбинации с другими препаратами. Иногда исследования III фазы продолжаются, когда документы на регистрацию в соответствующий регуляторный орган уже поданы. В таком случае пациенты продолжают получать препарат, пока он не

будет зарегистрирован и не поступит в продажу. Исследования подобного рода классифицируются иногда как фаза III.

Фаза IV клинического испытания. Уточняется безопасность и эффективность в пределах показаний в которых разрешено медицинское применение. Изменение сроков лечения и схем дозирования. Применение у новых возрастных групп. Экономическая активность применения. Результаты длительного применения лекарственного средства для выявления зависимости. Фаза IV также известна как пострегистрационные исследования. Это исследования, проводимые после регистрации препарата в соответствии с утверждёнными показаниями. Это исследования, которые не требовались для регистрации препарата, однако необходимы для оптимизации его применения. Требование о проведении этих исследований может исходить как от регуляторных органов, так и от компании-спонсора. Целью этих исследований может быть, например, завоевание новых рынков для препарата (например в случае, если препарат не изучался на предмет взаимодействия с другими препаратами). Важная задача фазы IV – сбор дополнительной информации по безопасности препарата на достаточно большой популяции в течение длительного времени [1].

1.2 Процесс разработки нового фармакологически активного вещества

Процесс разработки нового фармакологически активного вещества представляет собой:

- 1) Поиск соединения – лидера
- 2) Оптимизация соединения – лидера
- 3) Улучшение фармакокинетических и фармацевтических свойств

Соединение – лидер – вещество которое представляет собой особый интерес в плане его дальнейшего продвижения как возможного кандидата в лекарственные средства. На рисунке 1.1 изображён структурный прототип лекарства. Вначале задача создания лекарства сводится к тому, чтобы синтезировать его прототип если он не был найден классическим скринингом.

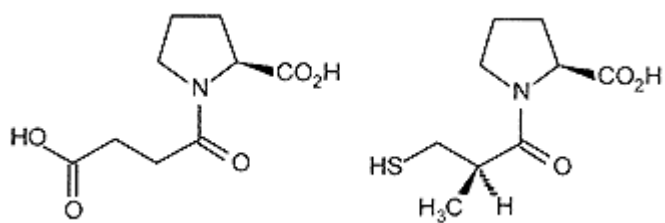


Рисунок 1.1 – N-сукцинил-пролин – пример соединения-лидера во время создания химического препарата каптоприла, снижающего кровяное давление

При драг-дизайне принято использовать правило пяти (Правило Липински):

- 1) структуры должны иметь менее пяти атомов-доноров водородной связи;
- 2) обладать молекулярным весом менее 500;
- 3) иметь липофильность ($\log P$ – коэффициент распределения вещества на границе раздела вода-октанол) менее 5 или логарифм коэффициента октанол/вода;
- 4) иметь суммарно не более 10 атомов азота и кислорода.

Исключения могут составлять вещества имеющие специфические переносчики: антибиотики, фунгицидные и протиропротозойные средства, витамины или сердечные гликозиды. В таблице 1.2 указаны примеры и исключения из Провила Липински.

Таблица 1.2 – Примеры соединений по правилу Липински и исключения

Вещество	$\lg P$	ОН + NH	M	N+O
Ацикловир	-0,09	4	225,21	8
Каптоприл	0,64	1	217,29	4
Циметидин	0,82	3	252,34	6
Диазепам	3,36	0	284,75	3
Дексаметазон	1,85	3	392,47	5
Диклофенак	3,99	2	296,15	3
Пироксикам	0,00	2	331,35	7
Эритромицин	-0,14	5	773,95	14
Циклоспорин	-0,32	5	1202,64	23

В таблице 1.3 Указаны основные химические свойства соединения-лидера

Таблица 1.3 – Химические свойства соединения-лидера

Свойство	Характеристика
Активность	Высокое сродство к мишени
Селективность	Низкое сродство к родственным мишеням
Биология	Проявление активности в клеточных пробах
Структура	Возможность получения аналогов
Стабильность	Химически стабильно
Хиральность	Индивидуальный энантиомер (для хиральных веществ)
Патентный статус	Ранее не запатентовано
Метаболизм	Высокая метаболическая стабильность
Энзимология	Не является потенциальным ингибитором (вещество, замедляющее протекание ферментативной реакции) или индуктором ферментов
Биодоступность	Высокая и дозозависимая
Проницаемость	Хорошо проникает через клеточную мембрану
Транспорт	Не ингибирует с Р-гликопротеин
Токсичность	Нецитотоксично и немутагенно
Безопасность	Не влияет активность hERG-каналов в сердце

Способы поиска соединения лидера:

1) эмпирический:

- а) случайные открытия;
- б) природные соединения растения;
- в) биохимические процессы;
- г) побочное действие лекарственного средства;
- д) классический скрининг;

2) эволюционный:

- а) комбинаторный синтез;
- б) комбинаторное моделирование, докинг;

- в) создание клонов;
- г) генные технологии.

Примером случайного открытия соединения лидера может служить открытый Барнеттом Розенбергом препарат цисплатин во время опытов с платиной. Во время электрохимической коррозии комплексные соединения платины вызывали нарушения деления и гибель клетки кишечной палочки.

Используя классический скрининг, было обнаружен препарат сальварсан, в составе которого присутствует мышьяк. Этот препарат применялся при заболевании сифилисом как в ранней, так и поздней стадии в начале XX века. Также классическим скринингом Герхардт Домагком было выявлено, что пронтозил обладает антимикробным действием по отношению к большому количеству микроорганизмов, за что и получил в 1939 году Нобелевскую премию [2].

Комбинаторный синтез – применение комбинаторных процессов для получения наборов соединений из большого числа одноступенчатых реагентов.

Скаффолд – центральная часть молекулы, общая для всех членов библиотеки (рисунок 1.2).

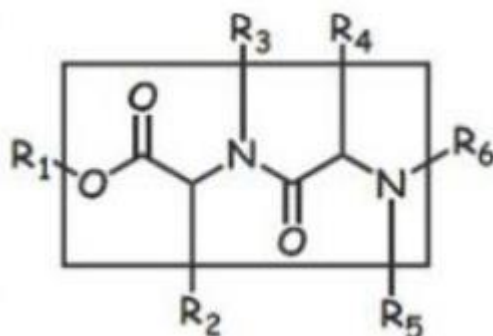


Рисунок 1.2 – Пример скаффолда

Используются комбинаторные библиотеки для поиска и оптимизации соединения-лидера. Для поиска используются обширные библиотеки, множество биологических мишеней, твердофазный синтез, а также скрининг смесей с последующей расшифровкой активных структур.

Однако для оптимизации лидера после нахождения прототипа используются целевые библиотеки, определённые структуры, выбирается одна биологическая мишень или группа родственных мишеней, производится синтез в растворе, автоматический параллельный синтез, а на выходы проводится оптимизированный скрининг индивидуальных соединений для оптимизации необходимых свойств.

Комбинаторная библиотека – набор соединений, полученных с использованием комбинаторного синтеза (рисунок 1.3).

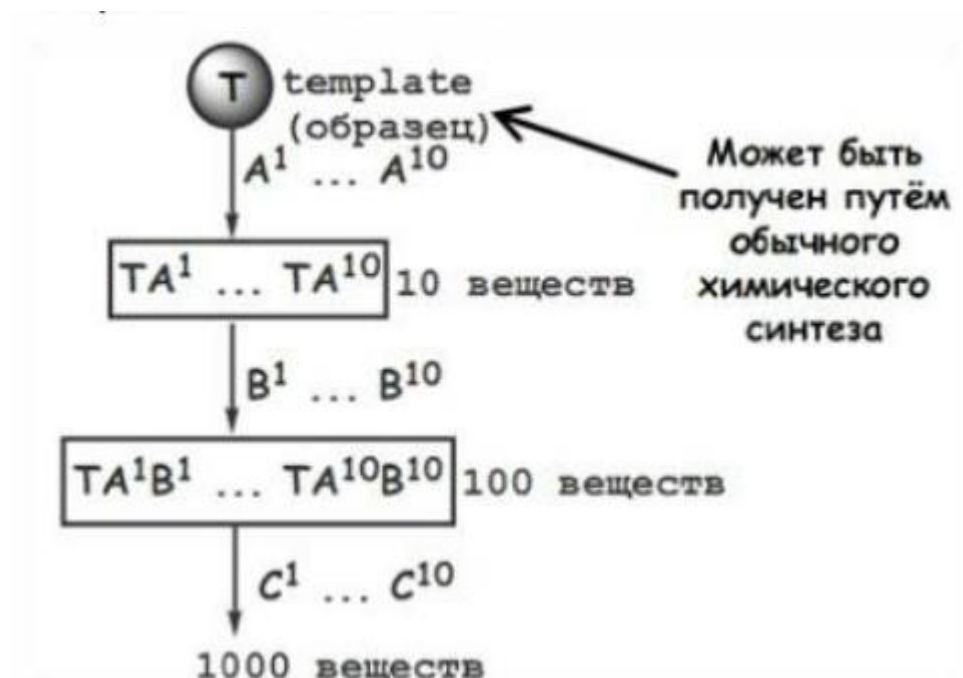


Рисунок 1.3 – Создание комбинаторной библиотеки

Хит – компонент библиотеки, активность которого превосходить предопределённый статистически значимый порог [3].

1.3 Высокопроизводительный скрининг

Высокопроизводительный скрининг – флуоресцентная хемилюминесцентная идентификация активных соединений, содержащей 96 ячеек ёмкостью 0,1мл.

Скрининг на грануле – взаимодействие с мишенями, мечеными ферментами, флуоресцирующей метой радионуклидами или хромофорами. Положительная реакция регистрируется по флуоресценции или изменению окраски. Активные гранулы могут быть отсортированы с последующим установление активного соединения.

Скрининг – оптимизированная конвейеризированная процедура, в результате которой большое количество химических соединений (>10000) проверяется на активность по отношению к специальной тестовой (имитирующей биологическую) системе:

- 1) низкопроизводительный (10000-50000 соединений);
- 2) среднепроизводительный (50000-100000 соединений);

3) высокопроизводительный (100000–5000000 соединений).

Одна из задач, которая стоит для фармакологии это избежание и оптимизация скрининга, а целенаправленный поиск структур с необходимыми свойствами. Одно из решений – реакции с помощью биомишеней.

Цель – найти лекарственное средство, которое реагирует с биомишенью и восстанавливает её состояние. Список биомишеней публикуется государством. Этот перечень служит ориентиром для организаций, занимающихся разработкой лекарственных препаратов. Наиболее часто встречающиеся биомишени – это рецепторы и ферменты.

Один из самых ранних и самых важных этапов драг-дизайна – выбрать правильную мишень, воздействуя на которую можно специфическим образом регулировать одни биохимические процессы и, по возможности, не затрагивая при этом другие. Однако, как уже было сказано, такое не всегда возможно: далеко не все заболевания являются следствием дисфункции только одного белка или гена.

Кроме того, при выборе мишени не следует забывать о таком явлении, как полиморфизм – то есть о том, что ген может существовать в разных изоформах у разных популяций или рас людей, что приведет к разному эффекту лекарства на разных больных.

В отсутствие лиганда рецептор характеризуется собственным уровнем клеточного ответа – так называемой базальной активностью. По типу модификации клеточного ответа лиганды делят на три группы:

Агонисты увеличивают клеточный ответ. Нейтральные агонисты связываются с рецептором, но не изменяют клеточный ответ по сравнению с базальным уровнем. Обратные агонисты, или антагонисты понижают клеточный ответ (рисунок 1.4) [4].

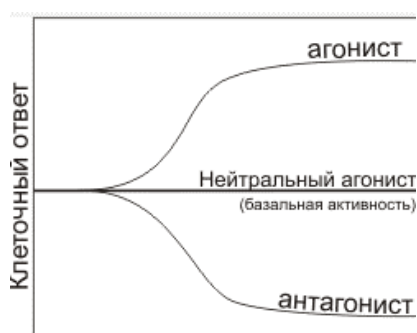


Рисунок 1.4 – Три типа влияния лигандов на клеточный ответ: увеличение ответа (положительный агонист), постоянство ответа, но конкуритрование за связывании с другими лигандами (нейтральный агонист) и уменьшение ответа (антагонист).

Для скрининга с использованием биомиметической очень критична эффективность, стоимость и время, необходимое на операцию. Скрининг производится на специальных установках (рисунок 1.5), способных работать без остановки в течении долгого времени.

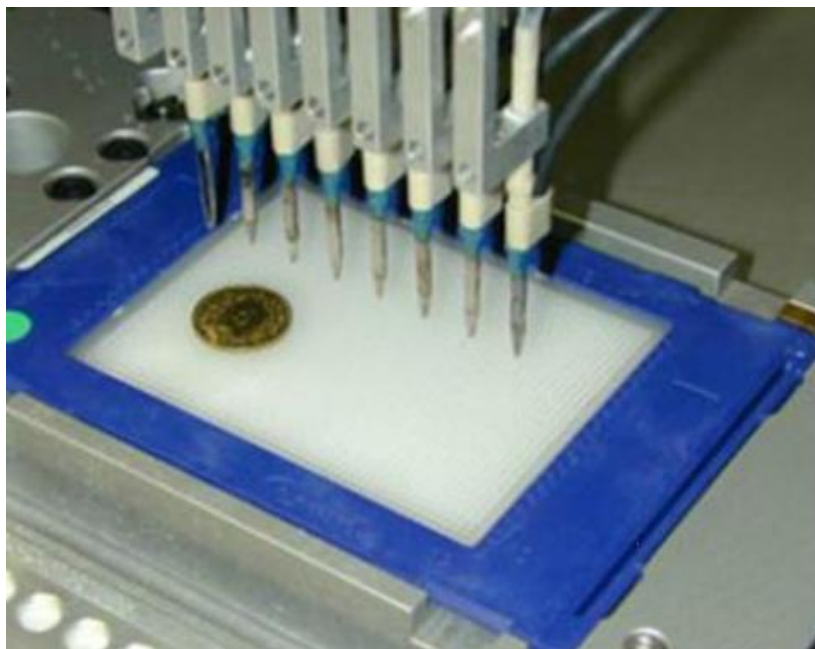


Рисунок 1.5 – Пример аппаратуры: роботизированная пипетка

В промышленных условиях используют такие средства как роботизированная пипетка, в автоматическом режиме наносящая образцы тестируемых соединений в плашку с системой для скрининга. Типичное количество углублений в плашке – тысячи единиц. Объём системы в одной лунке – микролитры. Объём образца – нанолитры.

Принцип скрининга: в плашки, которые содержат тестовую систему, робот капает из пипетки исследуемые вещества, следуя заданной программе.

Далее детектор считывает данные различным способом: радиоактивным сигналом, флуоресценцией, биолуминесценцией, поляризацией излучения и так далее. Детектор считывает пипетки с биологической активностью. В результате количество тестируемых соединений сокращается на 3-4 порядка. Соединения, для которых в процессе скрининга выявлена активность выше базальной, называются прототипами [5].

1.4 Молекулярный докинг

Молекулярный докинг – процесс стыковки соединения к рецептору с целью поиска наиболее выгодных положений и выявления факторов, изменение которых может привести к изменению их взаимодействия.

Основная цель докинга – построение модели структуры комплекса молекулы к рецептору(биомишени) Докинг также применяется при определении функциональных особенностей поверхности мишени и особенности взаимодействия молекул в комплексе, при поиске соединений к конкретной мишени среди большого количества молекул, а также при синтезе не существующих ранее соединений к конкретной мишени. Методы докинга подразделяются на жёсткие и гибкие.

Современные алгоритмы докинга:

- 1) фрагментальный докинг;
- 2) генетический алгоритм;
- 3) метод монте карло;
- 4) применение решёток аппроксимации;
- 5) тотальный перебор вариантов (скрининг).

Существуют два подхода при моделировании докинга (стыковки). Один подход использует технику соответствия, которая описывает белок и лиганд как дополнительные поверхности. Второй подход моделирует фактический процесс стыковки, в котором вычисляются попарные энергии взаимодействия. У обоих подходов есть существенные преимущества, а также некоторые ограничения. Геометрическое соответствие (методы взаимозависимости формы) описывается для белка и лиганда как ряд особенностей, которые позволяют их стыковать. Эти особенности могут включать как саму молекулярную поверхность, так и описание дополнительных особенностей поверхности. В этом случае молекулярная поверхность рецептора описывается с точки зрения её доступности площади поверхности для растворителя, а молекулярная поверхность лиганда описывается с точки зрения её соответствия описанию поверхности рецептора. Взаимозависимость между двумя поверхностями составляет описание соответствия формы, которое может помочь обнаружению дополнительной позы стыковки цели и молекул лиганда. В другом подходе нужно описать гидрофобные особенности белка, используя повороты в атомах главной цепи.

При симулировании белок и лиганд отделены некоторым физическим расстоянием, и лиганд находит своё положение в активный сайт белка после определенного числа «шагов». Шаги включают преобразования твердого тела, такие как перемещение и вращение, а также внутренние изменения структуры

лиганда включая угловые вращения. Каждый из этих шагов в пространстве изменяет полную энергетическую оценку системы, и, следовательно, она вычисляется после каждого движения. Очевидное преимущество этого метода состоит в том, что это позволяет исследовать гибкость лиганда во время моделирования, тогда как методы взаимозависимости формы должны использовать некоторые другие методы, чтобы узнавать о гибкости лиганда. Другое преимущество состоит в том, что процесс физически ближе к тому, что происходит в действительности, когда белок и лиганд приближаются к друг другу после молекулярного распознавания (рисунок 1.6).

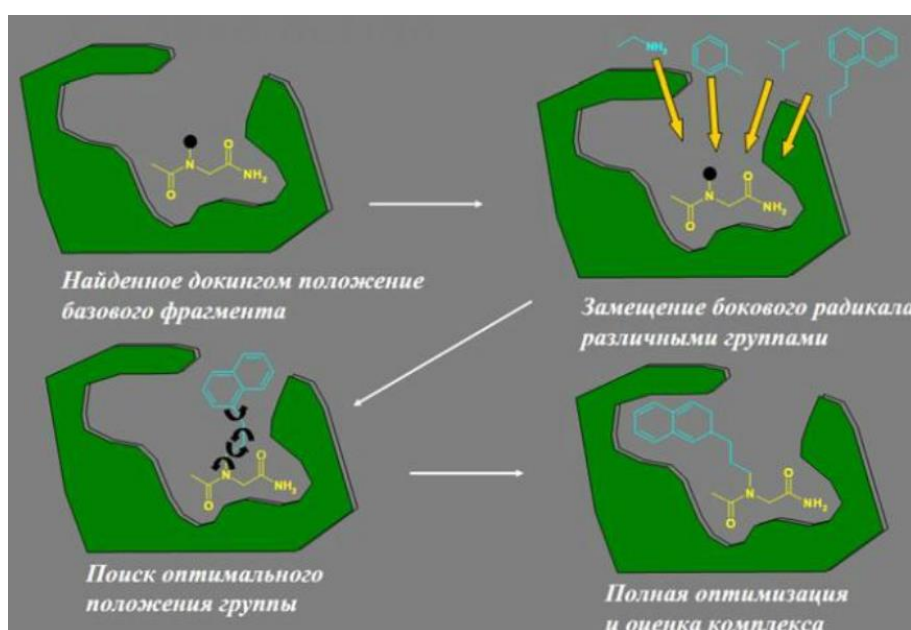


Рисунок 1.6 – Процесс фрагментального докинга

Основной принцип всех алгоритмов – нахождение глобального минимума энергии взаимодействия между соединением (лигандом) и рецептором.

В алгоритмах докинга в качестве стартовой информации используют пространственную структуру белка-рецептора и структуру лиганда, расположение которого относительно активного центра (скаффолда) оптимизируется в процессе докинга. Результатом докинга является набор расположений лиганда, взаимодействующих с белковой субстанцией связывания наилучшим образом, с точки зрения оценочной функции докинга.

Подобные эмпирические оценочные функции отражают различные аспекты межмолекулярных взаимодействий в комплексе белок-лиганд. Здесь можно провести аналоги между оценочной функцией и свободной энергией связывания лиганда белком. Существующие современные программы докинга

используют разные подходы для поиска конформаций лиганда (эвристический поиск с локальной минимизацией) и их проверки с помощью оценочных функций (модифицированное силовое поле AMBER и GoldScore/ChemScore соответственно).

Существует много программ для теоретической стыковки белков. Одна из них приведена на рисунке 1.7. Большая часть работает по следующему принципу: один белок фиксируется в пространстве, а второй поворачивается вокруг него разнообразными способами. При этом, для каждой конфигурации поворотов производятся оценочные расчеты по оценочной функции. Оценочная функция основана на поверхностной комплементарности, электростатических взаимодействиях, Ван-дер-Ваальсовском отталкивании и так далее. Проблема при этом поиске в том, что вычисления по всему конфигурационному пространству требуют много времени на вычисления, редко приводя к единственному решению.

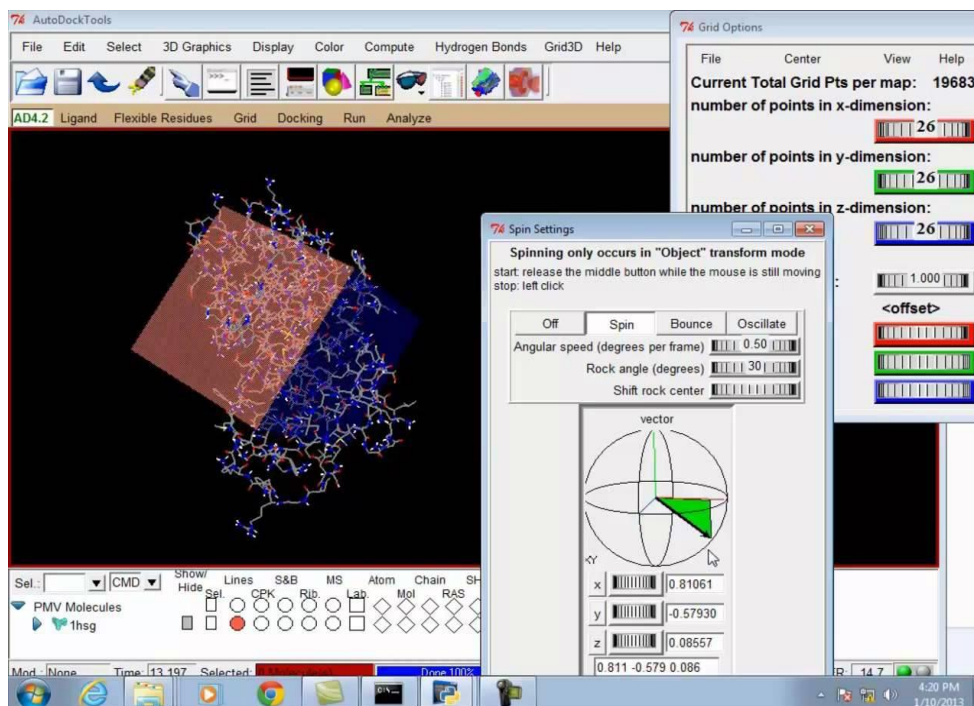


Рисунок 1.7 – ПО предназначенное для автоматизированного молекулярного докинга

Расчёт значений оценочной функции должен занимать минимальное компьютерное время, так как при высокопроизводительном докинге исследуется взаимодействие до миллиона соединений с белком-мишенью. Для интенсификации процесса скрининга в оценочные функции, используемые в программах докинга, часть включают упрощённые эмпирические термы.

Значения оценочных функций, как правило, слабо коррелируют с экспериментально определяемыми значениями свободной энергии связывания. Более того, во многих случаях комплексные белок-мишень, полученные методом докинга, характеризуется худшими значениями оценочной функции по сравнению с заведомо неверными комплексами. В связи с этим при высокопроизводительном докинге, во время отбора только одного комплекса с лучшим значением оценочной функции, есть риск потерять перспективное соединение – прототип вещества, так что необходимы дополнительные критерии и экспериментальная информация для выбора наиболее корректных результатов докинга.

Необходимо обратить внимание на необходимость критически анализировать результаты применения методов молекулярного моделирования в каждой конкретной задаче. Нужно убедиться, что используемые методы и протоколы приводят к корректным, не противоречащим экспериментальным результатам. Если результаты моделирования показывают согласие с экспериментальными данными, то есть основания использовать их для предсказания свойств трехмерных моделей изучаемых белков [6].

2 АНАЛИЗ ПРОТОТИПОВ И ПОСТАНОВКА ЗАДАЧИ

2.1 Задачи на проектирование

В рамках данного проекта необходимо разработать веб-приложение, которое позволит определить терапевтический класс химического соединения, используя методы машинного обучения используя и специальный способ ввода формул.

Для разработки программного продукта были поставлены следующие задачи:

- 1) возможность распознавать химическую формулу;
- 2) конвертация графического представления в машинный формат;
- 3) вывод терапевтического класса.

2.2 Использование системы

Для работы с программой необходимо перейти по заданному URL адресу. Пользователя приветствует редактор химических формул (рисунок 2.1).

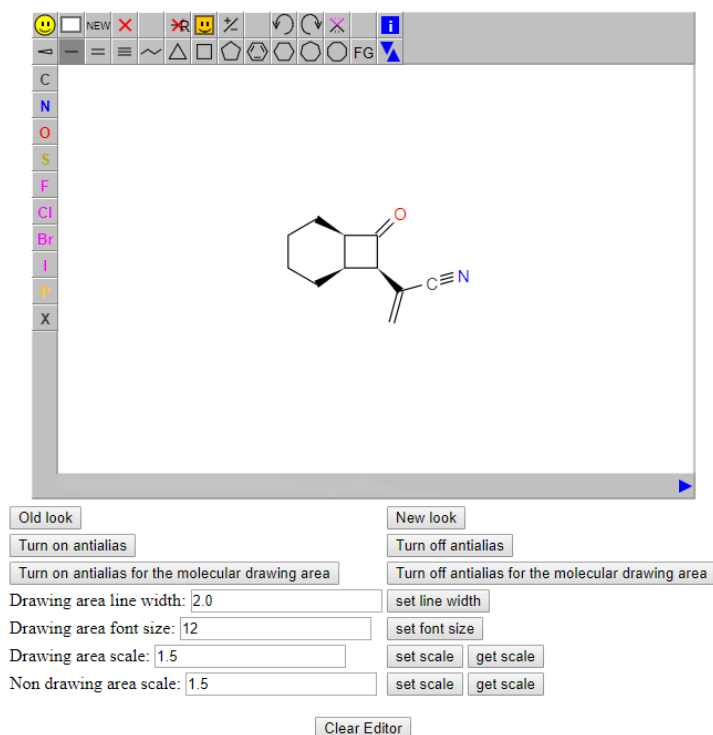


Рисунок 2.1 – Программный интерфейс

Редактор состоит из 2 панелей – верхней в два ряда и боковой левой.

Боковая левая панель состоит из набора атомов для быстрого доступа.

Верхнюю панель рассмотрим подробнее

Элементы управления:



– показывает молекулу SMILES в строковом представлении;



– очистка области редактирования;



– добавление нового компонента молекулы;



– удаление связи или атома;



– удаление функциональной группы;



– запуск нумерации атомов или отображение реакций;



– построение формулы с помощью SMARTS;



– переключатели между заряженными состояниями на атоме;



– вход реакции;



– отмена действия;



– возврат отмены;



– информацию о версии ;



– добавление входной стереосвязи ;



– цепной инструмент, позволяет создавать цепи и кольца;



– позволяет вводить неорганические атомы или атомы в нестандартном валентном состоянии ;



– всплывающее меню функциональной группы ;



– переместить выбранный атом при рисовании переполненных структур.

После проектирования формулы необходимо получить текстовое представление формулы нажатием кнопки “Get smiles” и после этого нажать на кнопку “Predict therapy” (рисунок 2.2).

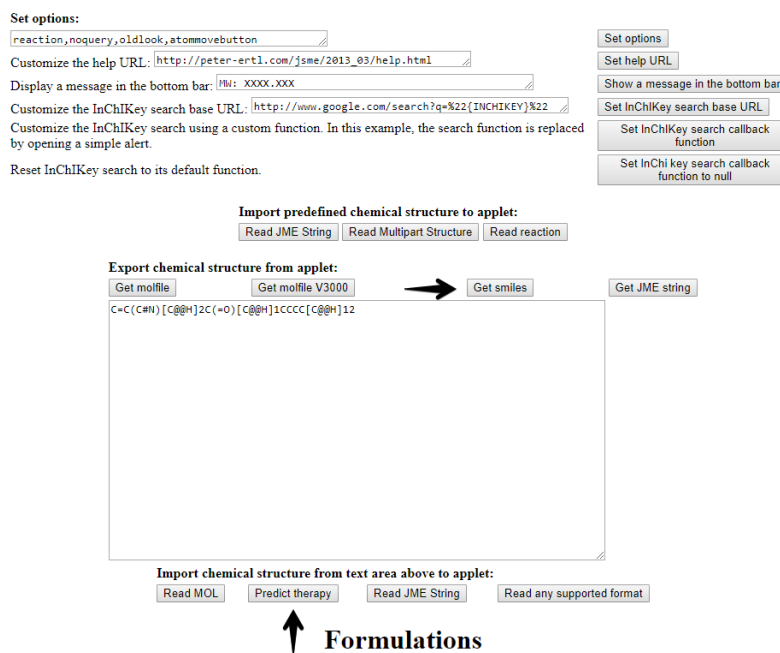


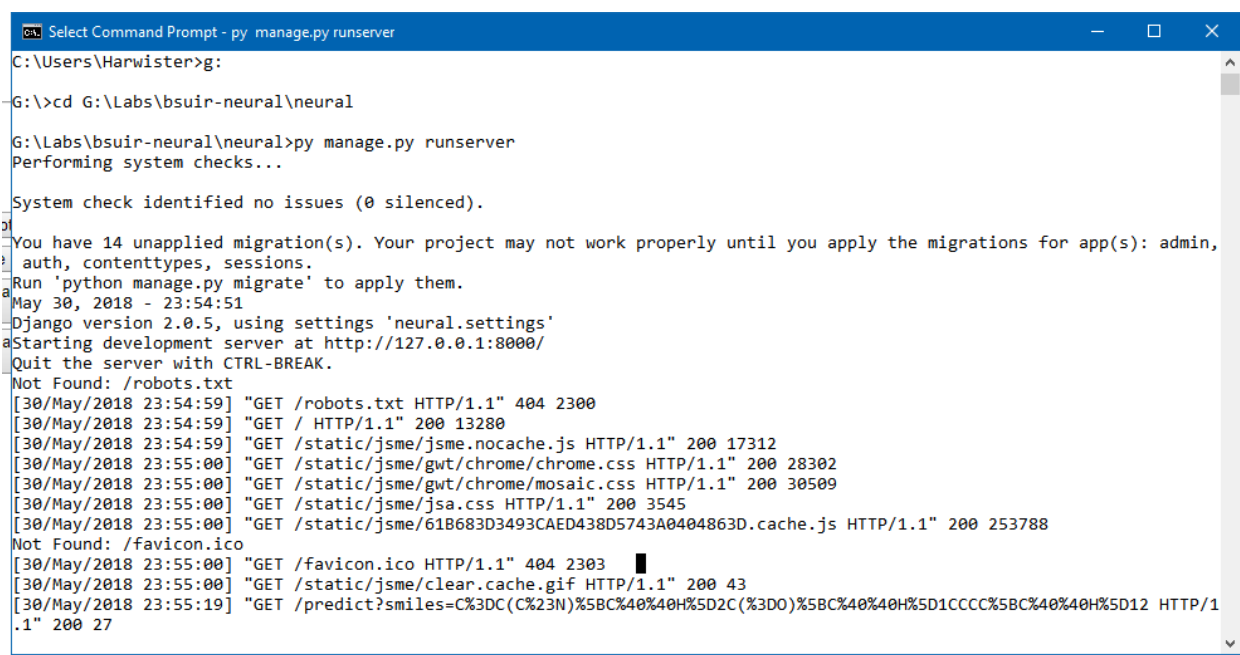
Рисунок 2.2 – Второй этап работы с программой.

База терапевтических классов состоит из семнадцати различных категорий:

- 1) Пищеварительный тракт и продукты обмена.
- 2) Класс крови и свертывания крови.
- 3) Сердечно-сосудистый класс.
- 4) Дерматологический класс.
- 5) Композиционный класс.
- 6) Мочеполовой класс (и половые гормоны)».
- 7) Гормональный (исключая половые гормоны) класс».
- 8) Иммунологический класс.
- 9) Антиинфекционный класс.
- 10) Онкологический класс.
- 11) Мышечно-скелетный класс.
- 12) Неврологический класс.
- 13) Противопаразитарный класс.
- 14) Респираторный класс.
- 15) Сенсорный класс.
- 16) Биотехнологии.
- 17) Другое.

2.3 Обобщённый алгоритм работы программы

Приложение начинает работу с запуска встроенного веб-сервера Django. Консольный вывод приведён на рисунке 2.3.



```
Select Command Prompt - py manage.py runserver
C:\Users\Harwister>g:
G:\>cd G:\Labs\bsuir-neural\neural
G:\Labs\bsuir-neural\neural>py manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
You have 14 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin,
auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
May 30, 2018 - 23:54:51
Django version 2.0.5, using settings 'neural.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
Not Found: /robots.txt
[30/May/2018 23:54:59] "GET /robots.txt HTTP/1.1" 404 2300
[30/May/2018 23:54:59] "GET / HTTP/1.1" 200 13280
[30/May/2018 23:54:59] "GET /static/jsme/jsme.nocache.js HTTP/1.1" 200 17312
[30/May/2018 23:55:00] "GET /static/jsme/gwt/chrome/chrome.css HTTP/1.1" 200 28302
[30/May/2018 23:55:00] "GET /static/jsme/gwt/chrome/mosaic.css HTTP/1.1" 200 30509
[30/May/2018 23:55:00] "GET /static/jsme/jsa.css HTTP/1.1" 200 3545
[30/May/2018 23:55:00] "GET /static/jsme/61B683D3493CAED438D5743A0404863D.cache.js HTTP/1.1" 200 253788
Not Found: /favicon.ico
[30/May/2018 23:55:00] "GET /favicon.ico HTTP/1.1" 404 2303
[30/May/2018 23:55:00] "GET /static/jsme/clear.cache.gif HTTP/1.1" 200 43
[30/May/2018 23:55:19] "GET /predict?smiles=C%3DC(C%23N)%5BC%40%40H%5D2C(%3DO)%5BC%40%40H%5D1CCCC%5BC%40%40H%5D12 HTTP/1.1" 200 27
```

Рисунок 2.3 – Консольный вывод сервера

После получения страницы с редактором и построения формулы приложение принимает принимает AJAX-запрос по URL адресу /predict с параметром smiles, содержащим текстовое представление формулы SMILES в каноническом виде.

Далее происходит подготовка данных для обучения с помощью алгоритма дерева решений. Все формулы из подготовленного набора приводятся к каноническому виду и векторизуются в массив из чисел с найденными различиями с помощью алгоритма CountVectorizer для передачи в дерево.

Тот же процесс проводится для параметра, спроектированного пользователем. После отработки процесса дерева решений номер категории передаётся в словарь категорий для перевода в строку. Ответ AJAX-запроса состоит из строкового названия категории и выводится в HTML-тег therapy.

3 ТЕХНОЛОГИИ И ПАТТЕРНЫ, ИСПОЛЬЗУЕМЫЕ ПРИ РАЗРАБОТКЕ

3.1 Python

В связи с наблюдаемым в настоящее время стремительным развитием персональной вычислительной техники, происходит постепенное изменение требований, предъявляемых к языкам программирования. Все большую роль начинают играть интерпретируемые языки, поскольку возрастающая мощь персональных компьютеров начинает обеспечивать достаточную скорость выполнения интерпретируемых программ. А единственным существенным преимуществом компилируемых языков программирования является создаваемый ими высокоскоростной код. Когда скорость выполнения программы не является критичной величиной, наиболее правильным выбором будет интерпретируемый язык, как более простой и гибкий инструмент программирования.

Отличительные характеристики языка: очень низкий порог вхождения, уже после одного дня изучения можно начать писать простые программы, минималистичный язык, с небольшим количеством конструкций, краткий код, прекрасно подходит для создания программ-обёрток, поддерживается импорт Си-библиотек, существует большое количество реализаций: CPython (основная реализация); Jython (реализация для JVM); IronPython (CLR); PyPy, очень хорошая поддержка математических вычислений (библиотеки NumPy, SciPy), используется для обработки естественных языков (NLTK), большое количество развитых web-фреймворков (Django, TurboGear, CherryPy, Flask).

Качество программного обеспечения: для многих основное преимущество языка Python заключается в удобочитаемости, ясности и более высоком качестве, отличающими его от других инструментов в мире языков программирования. Программный код на языке Python читается легче, а значит, многократное его использование и обслуживание выполняется гораздо проще, чем использование программного кода на других языках сценариев. Единообразие оформления программного кода на языке Python облегчает его понимание даже для тех, кто не участвовал в его создании. Кроме того, Python поддерживает самые современные механизмы многократного использования программного кода, каким является объектно-ориентированное программирование (ООП).

Высокая скорость разработки: по сравнению с компилируемыми или строго типизированными языками, такими как C, C++ и Java, Python во много раз повышает производительность труда разработчика. Объем программного кода на языке Python обычно составляет треть или даже пятую часть эквивалентного программного кода на языке C++ или Java. Это означает меньший объем ввода с клавиатуры, меньшее количество времени на отладку

и меньший объем трудозатрат на сопровождение. Кроме того, программы на языке Python запускаются сразу же, минуя длительные этапы компиляции и связывания, необходимые в некоторых других языках программирования, что еще больше увеличивает производительность труда программиста.

Переносимость программ: большая часть программ на языке Python выполняется без изменений на всех основных платформах. Перенос программного кода из операционной системы Linux в Windows обычно заключается в простом копировании файлов программ с одной машины на другую. Более того, Python предоставляет массу возможностей по созданию переносимых графических интерфейсов, программ доступа к базам данных, веб-приложений и многих других типов программ. Даже интерфейсы операционных систем, включая способ запуска программ и обработку каталогов, в языке Python реализованы переносимым способом.

Библиотеки поддержки: в составе Python поставляется большое число собранных и переносимых функциональных возможностей, известных как стандартная библиотека. Эта библиотека предоставляет массу возможностей, востребованных в прикладных программах, начиная от поиска текста по шаблону и заканчивая сетевыми функциями. Кроме того, Python допускает расширение как за счет ваших собственных библиотек, так и за счет библиотек, созданных сторонними разработчиками. Из числа сторонних разработок можно назвать инструменты создания веб-сайтов, программирование математических вычислений, доступ к последовательному порту, разработку игровых программ и многое другое. Например, расширение NumPy позиционируется как свободный и более мощный эквивалент системы программирования математических вычислений Matlab.

Интеграция компонентов: сценарии Python легко могут взаимодействовать с другими частями приложения благодаря различным механизмам интеграции. Эта интеграция позволяет использовать Python для настройки и расширения функциональных возможностей программных продуктов. На сегодняшний день программный код на языке Python имеет возможность вызывать функции из библиотек на языке C/C++, сам вызываться из программ, написанных на языке C/C++, интегрироваться с программными компонентами на языке Java, взаимодействовать с такими платформами, как COM и .NET, и производить обмен данными через последовательный порт или по сети с помощью таких протоколов, как SOAP, XML-RPC и CORBA.

С точки зрения функциональных возможностей Python можно назвать гибридом. Его инструментальные средства укладываются в диапазон между традиционными языками сценариев (такими как Tcl, Scheme и Perl) и языками разработки программных систем (такими как C, C++ и Java). Python обеспечивает простоту и непринужденность языка сценариев. Превышая возможности других языков сценариев, такая комбинация делает Python

удобным средством разработки крупномасштабных проектов. Ниже приводится список основных возможностей, которые есть в арсенале Python:

Динамическая типизация: Python сам следит за типами объектов, используемых в программе, благодаря чему не требуется писать длинные и сложные объявления в программном коде. В действительности, в языке Python вообще отсутствуют понятие типа и необходимость объявления переменных. Так как программный код на языке Python не стеснен рамками типов данных, он автоматически может обрабатывать целый диапазон объектов.

Автоматическое управление памятью: python автоматически распределяет память под объекты и освобождает ее ("сборка мусора"), когда объекты становятся ненужными. Большинство объектов могут увеличивать и уменьшать занимаемый объем памяти по мере необходимости.

Модульное программирование: для создания крупных систем Python предоставляет такие возможности, как модули, классы и исключения. Они позволяют разбить систему на составляющие, применять ООП для создания программного кода многократного пользования и элегантно обрабатывать возникающие события и ошибки.

Встроенные типы объектов: Python предоставляет наиболее типичные структуры данных, такие как списки, словари и строки, в виде особенностей, присущих самому языку программирования. Эти типы отличаются высокой гибкостью и удобством. Например, встроенные объекты могут расширяться и сжиматься по мере необходимости, могут комбинироваться друг с другом для представления данных со сложной структурой.

Встроенные инструменты: для работы со всеми этими типами объектов в составе Python имеются мощные и стандартные средства, включая такие операции, как конкатенация (объединение коллекций), получение срезов (извлечение части коллекции), сортировка, отображение и многое другое.

Библиотеки утилит: для выполнения более узких задач в состав Python также входит большая коллекция библиотечных инструментов, которые поддерживают практически все, что только может потребоваться,—от поиска с использованием регулярных выражений до работы в сети. Библиотечные инструменты языка Python — это то место, где выполняется большая часть операций.

Утилиты сторонних разработчиков: Python — это открытый программный продукт и поэтому разработчики могут создавать свои предварительно скомпилированные инструменты поддержки задач, решить которые внутренними средствами невозможно.

Когда запускается программа, Python сначала компилирует исходный текст (инструкции в файле) в формат, известный под названием байт-код. Компиляция — это этап перевода программы, а байт-код — это низкоуровневое, платформонезависимое представление исходного текста программы.

Интерпретатор Python транслирует каждую исходную инструкцию в группы инструкций байт-кода, разбивая ее на отдельные составляющие. Такая трансляция в байт-код производится для повышения скорости—байт-код выполняется намного быстрее, чем исходные инструкции в текстовом файле.

Интерпретатор сохраняет байт-код для ускорения запуска программ. В следующий раз, когда вы попытаете запустить свою программу, Python загрузит файл .рус и минует этап компиляции—при условии, что исходный текст программы не изменялся с момента последней компиляции. Чтобы определить, необходимо ли выполнять перекомпиляцию, Python автоматически сравнит время последнего изменения файла с исходным текстом и файла с байт-кодом. Если исходный текст сохранялся на диск после компиляции, при следующем его запуске интерпретатор автоматически выполнит повторную компиляцию программы.

Если интерпретатор окажется не в состоянии записать файл с байт-кодом на диск, программа от этого не пострадает, байт-код будет сгенерирован в памяти и исчезнет по завершении программы.

Байт-код – это внутреннее представление программ на языке Python. По этой причине программный код на языке Python не может выполняться так же быстро, как программный код на языке C или C++. Обход инструкций выполняет виртуальная машина, а не микропроцессор, и чтобы выполнить байт-код, необходима дополнительная интерпретация, инструкции которого требуют на выполнение больше времени, чем машинные инструкции микропроцессора. С другой стороны, в отличие от классических интерпретаторов, здесь присутствует дополнительный этап компиляции—интерпретатору не требуется всякий раз снова и снова анализировать инструкции исходного текста. В результате Python способен обеспечить скорость выполнения где-то между традиционными компилирующими и традиционными интерпретирующими языками программирования.

Недостатки Python: низкое быстродействие Классический Python, как и многие другие интерпретируемые языки, не применяющие, например, JIT-компиляторы, имеют общий недостаток – сравнительно невысокую скорость выполнения программ. Сохранение байт-кода позволяет интерпретатору не тратить лишнее время на перекомпиляцию кода модулей при каждом запуске, в отличие, например, от языка Perl. Кроме того, существует специальная JIT-библиотека `psuso`, позволяющая ускорить выполнение программ (однако приводящая к увеличению потребления оперативной памяти). Эффективность `psuso` сильно зависит от архитектуры программы.

Существуют реализации языка Python, вводящие высокопроизводительные виртуальные машины (VM) в качестве бэк-энда компилятора. Примерами таких реализаций может служить PyPy, базирующийся на LLVM; более ранней инициативой является проект Parrot.

Ожидается, что использование ВМ типа LLVM приведёт к тем же результатам, что и использование аналогичных подходов для реализаций языка Java, где низкая вычислительная производительность в основном преодолена.

Множество программ/библиотек для интеграции с другими языками программирования предоставляют возможность использовать другой язык для написания критических участков [7].

3.2 Django

Django – свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC. Проект поддерживается организацией Django Software Foundation.

Сайт на Django строится из одного или нескольких приложений, которые рекомендуется делать отчуждаемыми и подключаемыми. Это одно из существенных архитектурных отличий этого фреймворка от некоторых других (например, Ruby on Rails). Один из основных принципов фреймворка – DRY. Это принцип разработки программного обеспечения, нацеленный на снижение повторения информации различного рода, особенно в системах со множеством слоёв абстрагирования.

Также, в отличие от других фреймворков, обработчики URL в Django конфигурируются явно при помощи регулярных выражений, а не выводятся автоматически из структуры моделей контроллеров.

Для работы с базой данных Django использует собственный ORM, в котором модель данных описывается классами Python, и по ней генерируется схема базы данных.

Архитектура Django похожа на «Модель-Представление-Контроллер» (MVC). Контроллер классической модели MVC примерно соответствует уровню, который в Django называется Представление (англ. View), а презентационная логика Представления реализуется в Django уровнем Шаблонов (англ. Template). Из-за этого уровневую архитектуру Django часто называют «Модель-Шаблон-Представление»

Первоначальная разработка Django, как средства для работы новостных ресурсов, достаточно сильно отразилась на его архитектуре: он предоставляет ряд средств, которые помогают в быстрой разработке веб-сайтов информационного характера. Так, например, разработчику не требуется создавать контроллеры и страницы для административной части сайта, в Django есть встроенное приложение для управления содержимым, которое можно включить в любой сайт, сделанный на Django, и которое может управлять сразу несколькими сайтами на одном сервере. Административное приложение позволяет создавать, изменять и удалять любые объекты

наполнения сайта, протоколируя все совершённые действия, и предоставляет интерфейс для управления пользователями и группами.

Django-приложение состоит из четырех основных компонентов (рисунок 3.1).

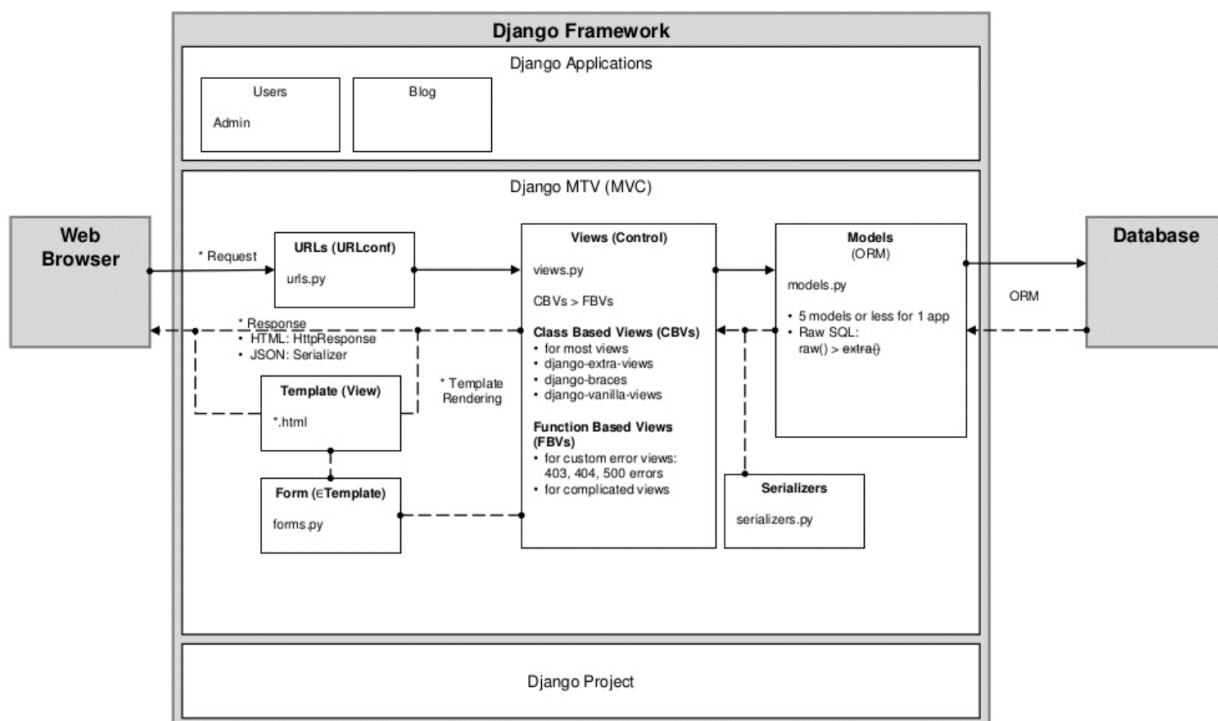


Рисунок 3.1 – Схема фреймворка Django

1) модель данных – данные являются сердцевиной любого современного Web-приложения. Модель–важнейшая часть приложения, которое постоянно обращается к данным при любом запросе из любой сессии. Любая модель является стандартным Python классом. Объектно-ориентированный маппер (ORM) обеспечивает таким классам доступ непосредственно к базам данных. Если бы не было ORM, программисту пришлось бы писать запросы непосредственно на SQL. Модель обеспечивает облегченный механизм доступа к слою данных, инкапсулирует бизнес-логику. Модель не зависит от конкретного приложения. Данными можно манипулировать даже из командной строки, не используя при этом Web-сервер;

2) представление (view) – выполняют разнообразные функции, в том числе контролируют запросы пользователя, выдают контекст в зависимости от его роли. View–это обычная функция, которая вызывается в ответ на запрос какого-то адреса (URL) и возвращает контекст;

3) шаблоны – являются формой представления данных. Шаблоны имеют свой собственный простой метаязык и являются одним из основных средств вывода на экран;

4) URL – механизм внешнего доступа к представлениям (view). Встроенные в URL регулярные выражения делают механизм достаточно гибким. При этом одно представление может быть сконфигурировано к нескольким урлам, предоставляя доступ различным приложениям. Здесь поддерживается философия закладок: URL становятся самодостаточными и начинают жить независимо от представления.

В дистрибутив Django также включены приложения для системы комментариев, синдикации RSS и Atom, «статических страниц» (которыми можно управлять без необходимости писать контроллеры и представления), перенаправления URL и другое [8].

3.3 HTML

Совершенствование технических возможностей средств вычислительной техники, развитие коммуникационных средств и технологий управления информационными ресурсами в последние годы привели к появлению более крупных информационных систем. Речь идет о масштабах систем не только относительно объема поддерживаемых информационных ресурсов, но и числа их пользователей. Объем информационных ресурсов Web в настоящее время исчисляется многими миллионами страниц.

В связи с этим развитием информационных технологий, сетей, а также информационных систем получил широкое распространение язык гипертекстовой разметки HTML. Информационные системы при этом рассматриваются как инструмент моделирования реальности, реализующей различные подходы. В последние годы стали появляться инструментальные средства и крупные информационные системы, в которых совместно используются различные информационные технологии. Сейчас существует множество специализированных программ для разработки Web сайтов. Такие программы, облегчают работу разработчикам в создании Web страниц со сложным дизайном, позволяют динамически генерировать страницы Web.

Для информационных технологий характерна деятельность по стандартизации различных аспектов. Такая деятельность направлена на обеспечение переносимости приложений и информационных ресурсов между различными программно-аппаратными платформами, повторное использование ресурсов, в частности это может быть использование программных компонентов приложений.

Информационные системы сегодня применяются во всех областях общественной жизни и научной деятельности. Курсовая работа предназначена

для обобщения накопленного отечественного и зарубежного опыта в разработке информационных систем связанная с Web-технологиями, выявление общих положений и принципов их построения и развития.

Представленная работа показывает значимость и эффективность использования информационных систем в первую очередь для поддержки человеческой деятельности в различных областях науки, образования и культуры. Популярность Internet во многом вызвана появлением WorldWideWeb (WWW), так как это первая сетевая технология, которая предоставила пользователю простой современный интерфейс для доступа к разнообразным сетевым ресурсам. Простота и удобство применения привели к росту числа пользователей WWW и привлекли внимание коммерческих структур. Далее процесс роста числа пользователей стал лавинообразным, и так продолжается до сих пор. На основе необходимости объединить все множество информационных ресурсов начала развиваться технология при помощи, которой определяется гипертекстовая навигационная система. Этой технологией стал язык HTML. Технология HTML на начальном этапе была чрезвычайно проста, и практически все пользователи сети одновременно получили возможность попробовать себя в качестве создателей и читателей информационных материалов, опубликованных во Всемирной паутине. Дело в том, что при разработке различных компонентов технологии предполагалось, что квалификация авторов информационных ресурсов и их оснащенность средствами вычислительной техники будут минимальными.

Язык HTML (HyperTextMarkupLanguage, язык разметки гипертекста) относится к числу так называемых языков разметки текста (markuplanguages). Под термином "разметка" понимается общая служебная информация, которая не выводится вместе с документом, но определяет; как должны выглядеть те или иные фрагменты документа. Например, вы можете потребовать, чтобы какое-либо слово выводилось жирным или курсивным шрифтом, вывести отдельный абзац особым шрифтом или оформлять заголовки увеличенным шрифтом.

В наши дни существует множество разных языков разметки. Например, в коммуникационных программах особая форма разметки определяет смысл каждого пакета из нулей и единиц, пересылаемого в Internet. Впрочем, любой язык разметки должен решать две важные задачи:

- 1) язык определяет синтаксис разметки;
- 2) язык определяет смысл разметки.

Наиболее распространенным из языков разметки Web-страниц является HTML. Это язык разметки был создан и рекламировался как одна из конкретизаций SGML. Впервые предложенный в 1974 году Чарльзом Голдфарбом и в дальнейшем после значительной доработки принятый в качестве официального стандарта ISO, SGML

(StandardGeneralizedMarkupLanguage, Стандартный обобщенный язык разметки) представляет собой метаязык – систему для описания других языков.

Появление стандарта SGML было обусловлено необходимостью совместного использования данных разными приложениями и операционными системами. Даже в далеких 60-х годах у пользователей компьютеров возникало немало проблем с совместимостью. Проанализировав недостатки многих нестандартных языков разметки, трое ученых из IBM – Чарльз Гольдфарб (Charles Goldfarb), Эд Мо-шер (Ed Mosher) и Рэй Лори (Ray Lorie) – сформулировали три общих принципа, обеспечивающих возможность совместной работы с документами в разных операционных системах:

- 1) использование единых принципов форматирования во всех программах, выполняющих обработку документов. Вполне логичное требование – всем нам хорошо известно, как трудно договориться между собой людям, говорящим на разных языках. Наличие единого набора синтаксических конструкций и общей семантики заметно упрощает взаимодействие между программами;

- 2) специализация языков форматирования. Благодаря возможности построения специализированного языка на базе набора стандартных правил программист перестает зависеть от внешних реализаций и их представлений о потребностях конечного пользователя;

- 3) четкое определение формата документа. Правила, определяющие формат документа, задают количество и маркировку языковых конструкций, используемых в документе. Применение стандартного формата гарантирует, что пользователь будет точно знать структуру содержимого документа. Обратите внимание: речь идет не о формате отображения документа, а о его структурном формате. Набор правил, описывающих этот формат, называется "определением типа документа" (document type definition, DTD).

HTML (HypertextMarkupLanguage, Язык разметки гипертекста) – это компьютерный язык, лежащий в основе WorldWideWeb. HTML основан на стандарте SGML гипертекстовый язык разметки документов для их представления в Web. Стандарты языка HTML, одного из ключевых стандартов Web, разрабатываются и поддерживаются консорциумом W3C. Основателем этого международного консорциума является Тим Бернес-Ли (TimBerners-Lee). Консорциум помимо создания стандартов форматирования, является центром разработки SemanticWeb (семантическая сеть). Средствами языка HTML обеспечивается форматная разметка документов, определяются гиперсвязи между документами и/или их фрагментами.

В качестве основы написания кода HTML был выбран обычный текстовый файл. Таким образом, гипертекстовая база данных в концепции WWW – это набор текстовых файлов, размеченных на языке HTML, который

определяет форму представления информации (разметка) и структуру связей между этими файлами и другими информационными ресурсами (гипертекстовые ссылки).

Разработчики HTML смогли решить две задачи: предоставить дизайнерам гипертекстовых баз данных простое средство создания документов и сделать это средство достаточно мощным, чтобы отразить имевшиеся на тот момент представления об интерфейсе пользователя гипертекстовых баз данных.

Первая задача была решена за счет выбора теговой модели описания документа. Язык HTML позволяет размечать электронный документ, который отображается на экране с полиграфическим уровнем оформления; результирующий документ может содержать самые разнообразные метки, иллюстрации, аудио- и видеофрагменты и так далее. В состав языка вошли развитые средства для создания различных уровней заголовков, шрифтовых выделений, различные списки, таблицы и многое другое.

Вторым важным моментом, повлиявшим на судьбу HTML, стало то, что в качестве основы был выбран обычный текстовый файл. Среда редактирования HTML является нейтральной полосой между простейшим текстовым файлом и приложением WYSIWYG (whatyouseeiswhatyouget – что вы видите, то и получаете). Выбор среды редактирования дает все преимущества текстового редактирования.

Гипертекстовые ссылки, устанавливающие связи между текстовыми документами, постепенно стали объединять самые различные информационные ресурсы, в том числе звук и видео. Система гиперссылок HTML позволяет построить систему взаимосвязанных документов по различным критериям. Язык HTML содержит команды (тэги), позволяющие управлять формой и размером шрифтов, размером и расположением иллюстраций, позволяет осуществлять переход от фрагмента текста или иллюстрации к другим html-документам – так называемую гипертекстовую ссылку. Документ в html-формате представляет собой текстовый файл, содержащий все необходимые сведения о выводимой на экран информации. Для управления сценариями просмотра страниц Website (гипертекстовой базы данных, выполненной в технологии World Wide Web) можно использовать языки программирования этих сценариев, например, JavaScript, Java и VBScript. Формы для введения пользователем данных, которые позднее подвергаются обработке и другую информацию можно обрабатывать с помощью специальных серверных программ (например, на языках PHP или Perl). Язык HTML позволяет помещать на страницы гипертекстовые ссылки и интерактивные кнопки, которые соединяют ваши Web-страницы с другими страницами того же Web-сайта, равно как и с другими Web-сайтами по всему миру.

HTML является языком разметки текста, а не языком программирования, который всего лишь один из инструментов (точнее, язык описания страниц), используемый при создании Web-страниц. В HTML ограничены возможности форматирования текста по сравнению с возможностями издательских программ, особенно при издании текста, насыщенного сложными элементами.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>HTML Document</title>
  </head>
  <body>
    <p>
      <b>
        Semi-bold text <i>italic</i>.
      </b>
    </p>
  </body>
</html>
```

Различают два вида html-документов – статические и динамические. Статические документы хранятся в файлах той файловой системы, которая используется web-сервером или браузером при просмотре локальных файлов. При размещении информации на web-сервере можно использовать динамические документы – такие, которые не существуют постоянно в виде файлов, а генерируются в момент запроса клиента. При этом для конечного пользователя не имеет значения динамический или статический способ представления документов.

Для генерирования динамического документа HTML требуется специально написанная программа по правилам, определяемым web-сервером. При планировании размещения информации на web-сервере, для правильного определения использования, какого-либо вида документов, необходимо учитывать степень обновляемости данных, их объем и частоту обращения.

3.4 Javascript

Язык программирования JavaScript разработан фирмой Netscape для создания интерактивных HTML-документов. Это объектно-ориентированный язык разработки встраиваемых приложений, выполняющих как на стороне

клиента, так и на стороне сервера. Синтаксис языка очень похож на синтаксис языка Java – поэтому его часто называют Java-подобным. Клиентские приложения выполняются браузером просмотра Web-документов на машине пользователя, серверные приложения выполняются на сервере.

При разработке обоих типов приложений используется общий компонент языка, называемый ядром и включающий определения стандартных объектов и конструкций (переменные, функции, основные объекты и средство LiveConnect взаимодействия с Java-апплетами), и соответствующие компоненты дополнений языка, содержащие специфические для каждого типа приложений определения объектов.

Клиентские приложения непосредственно встраиваются в HTML-страницы и интерпретируются браузером по мере отображения частей документа в его окне. Серверные приложения для увеличения производительности предварительно компилируются в промежуточный байт-код.

Основные области использования языка JavaScript при создании интерактивных HTML-страниц:

- динамическое создание документа с помощью сценария;
- оперативная проверка достоверности заполняемых пользователем полей форм HTML до передачи их на сервер;
- создание динамических HTML-страниц совместно с каскадными таблицами стилей и объектной моделью документа;
- взаимодействие с пользователем при решении “локальных” задач, решаемых приложением JavaScript, встроенном в HTML-страницу.

Как и любой другой язык программирования, JavaScript использует переменные для хранения данных определенного типа. Реализация JavaScript является примером языка свободного использования типов. В нем не обязательно задавать тип переменной. Ее тип зависит от типа хранимых в ней данных, причем при изменении типа данных меняется и тип переменной.

JavaScript поддерживает четыре простых типа данных:

- целый;
- вещественный;
- строковый;
- булевый, или логический.

Для присваивания переменным значений основных типов применяются литералы – буквенные значения данных соответствующих типов.

Целые литералы являются последовательностью цифр и представляют обычные целые числа со знаком или без знака:

123 // целое положительное число
-123 // целое отрицательное число
+123 // целое положительное число

Для задания вещественных литералов используется синтаксис чисел с десятичной точкой, отделяющей дробную часть числа от целой, или запись вещественных чисел в научной нотации с указанием после символа “e” или “E” порядка числа. Пример правильных вещественных чисел: “1.25”, “0.125e01”, “12.5E-1”, “0.0125E+2”

Строковый литерал – последовательность алфавитно-цифровых символов, заключенная в одинарные (‘) или двойные кавычки (“), например: “Ира”, ‘ИРА’. При задании строковых переменных нельзя смешивать одинарные и двойные кавычки. Недопустимо задавать строку, например, в виде “Ира’. Двойные кавычки – это один самостоятельный символ, а не последовательность двух символов одинарных кавычек. Если в строке нужно использовать символ кавычек, то строковый литерал необходимо заключать в кавычки противоположного вида:

“It’s a string” // Значение строки равно It’s a string

Булевы литералы имеют два значения: true и false, и используются для обработки ситуаций да/нет в операторах сравнения.

Каждая переменная имеет имя, которое должно начинаться с буквы латинского алфавита, либо символа подчеркивания “_”, за которым следует любая комбинация алфавитно-цифровых символов или символов подчеркивания. Следующие имена являются допустимыми именами переменных: Temp1, MyFunction, _my_Method.

Язык JavaScript чувствителен к регистру. Это означает, что строчные и прописные буквы алфавита считаются разными символами.

Определить переменную можно двумя способами:

- оператором var;
- оператором присваивания (=).

Оператор var используется как для задания, так и для инициализации переменной и имеет синтаксис:

var имя_переменной [= начальное_значение];

Необязательный оператор присваивания задает данные, которые содержит переменная. Их тип определяет и тип переменной. Например, следующий оператор:

var weekDay = “Пятница”;

Задаёт переменную `weekDay`, присваивает ей строковое значение “Пятница”, и тем самым определяет ее тип как строковый.

Если при определении переменной ей не присвоено никакого значения, то ее тип не определен. Ее тип будет определен только после того, как ей будет присвоено некоторое значение оператором присваивания `=`.

Весь набор управления языка можно разбить на три группы:

- операторы выбора, или условные;
- операторы цикла;
- операторы манипулирования с объектами.

К этой группе операторов относятся операторы, которые выполняют определенные блоки операторов в зависимости от истинности некоторого булевского выражения. Этот оператор условия `if...else` и переключатель `switch`.

Оператор условия `if` применяется, если необходимо вычислить некоторый блок операторов в зависимости от истинности заданного условия, и имеет следующий синтаксис:

```
if (условие) {  
    операторы1  
}  
[else {  
    операторы2  
}]
```

Первая группа операторов `операторы1` выполняется при условии истинности выражения `условие`. Необязательный блок `else` задает группу операторов `операторы2`, которая будет выполнена в случае ложности условия, заданного в блоке `if`.

Внутри группы выполняемых операторов могут использоваться любые операторы JavaScript, в том числе и операторы условия. Таким образом, можно создавать группу вложенных операторов условия и реализовывать сложные алгоритмы проверки.

В операторе `switch` вычисляется одно выражение и сравнивается со значениями, заданными в блоках `case`. В случае совпадения выполняются операторы соответствующего блока `case`. Синтаксис этого оператора следующий:

```
switch (выражение) {  
    case значение1 :  
    [операторы1]  
    break;
```

```

        case значение2 :
            [операторы2]
            break;
...
default :
    [операторы]
}

```

Если значение выражения в блоке switch равно значению 1, то выполняется группа операторов операторы 1, если равно значению 2, то выполняется группа операторов операторы 2 и т.д. Если значение выражения не равняется ни одному из значений, заданных в блоках case, то вычисляется группа операторов блока default, если этот блок задан, иначе происходит выход из оператора switch.

Необязательный оператор break, задаваемый в каждом из блоков case, выполняет безусловный выход из оператора switch. Если он не задан, то продолжается выполнение операторов в следующих блоках case до первого оператора break или до конца тела оператора switch.

Оператор цикла повторно выполняет последовательность операторов JavaScript, определенных в его теле, пока не выполнится некоторое заданное условие. В языке существует два оператора цикла: for и while. Они отличаются механизмом организации цикла.

Оператор цикла for позволяет организовать выполнение блока операторов заданное число раз. Он определяет переменную, называемую переменной цикла, которая изменяет свое значение во время выполнения цикла. Условие завершения цикла зависит от значения этой переменной. Оператор имеет следующий синтаксис:

```

for ([инициал_выражение];[условие];[изменяющее_выражение]) {
    [операторы]
}

```

Параметром инициал_выражение задается и инициализируется переменная цикла. Это выражение вычисляется один раз в начале выполнения цикла. После этого проверяется истинность выражения условие. Если оно истинно, то выполняется блок операторов тела цикла, ограниченного фигурными скобками; вычисляется изменяющее_выражение, содержащее переменную цикла, и снова проверяется истинность выражения условие. Если оно истинно, то повторяется цикл вычислений, если нет, то оператор цикла завершает свое выполнение.

Цикл `while` выполняется пока истинно выражение, задающее условие выполнения цикла. Его синтаксис, следующий:

```
while (условие) {  
    [операторы]  
}
```

Сначала проверяется истинность условия, заданного в заголовке цикла, а затем выполняются (или не выполняются) операторы тела цикла. Проверка истинности условия осуществляется на каждом шаге цикла. Использование этого цикла предполагает, что условное выражение окончания цикла меняется в зависимости от вычисленных значений переменных и выражений в теле цикла.

Иногда необходимо завершить цикл не по условию, задаваемому в заголовке цикла, а в результате вычисления некоторого условия в теле цикла. Для этой цели в JavaScript существуют операторы `break` и `continue`.

Оператор `break` завершает выполнение цикла и передает управление оператору, непосредственно следующим за оператором цикла. Оператор `continue` прекращает выполнение текущей итерации и начинает выполнение следующей, т.е. в цикле `while` он передает управление на проверку выражения условия цикла, а в цикле `for` – на вычисление выражения, изменяющее выражение.

Четыре оператора JavaScript предназначены для работы с объектами. Это оператор `new`, создающий новый объект, операторы `for...in` и `with` и ключевое слово `this`. Оператор `for...in` позволяет организовать цикл по свойствам объекта JavaScript. Синтаксис его следующий:

```
for (переменная_цикла in объект) {  
    [операторы]  
}
```

Этот цикл производит перебор свойств объекта. В переменной цикла на каждой итерации сохраняется значение свойства объекта. Количество итераций равно количеству свойств, существующих у заданного в заголовке цикла объекта.

Оператор `with` задает объект по умолчанию для блока операторов, определенных в его теле. Это означает, что все встречаемые в операторах этого блока свойства и методы, являются свойствами и методами указанного объекта. Применение данного оператора избавляет от необходимости указывать иерархию принадлежности объекта и сокращает исходный текст программы.

3.5 Ajax

AJAX – это коллекция технологий, существующих с момента появления Web. А вот и возможности, предоставляемые AJAX (как это представил Джис Джеймс Гаррет (Jesse James Garrett), он первым ввел термин «AJAX» для асинхронного JavaScript + XML):

- стандартно-базирующаяся презентация с использованием CSS;
- динамическое взаимодействие с использованием модели документа;
- взаимообмен данными с задействованием XML и XSLT;
- асинхронное извлечение данных с использованием XMLHttpRequest;
- JavaScript, связывающий все вместе.

Вкратце AJAX позволяет писать быстро реагирующие веб-приложения, в которых не нужно постоянно обновлять страницы. AJAX – простая технология, поддерживаемая всеми основными браузерами. Как можно вкратце отметить, единственным предварительным условием для внедрения AJAX является знание JavaScript.

AJAX – это подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. В результате при обновлении данных веб-страница не перезагружается полностью, и веб-приложения становятся более быстрыми и удобными.

AJAX – это не самостоятельная технология, а концепция использования нескольких смежных технологий. AJAX базируется на двух основных принципах:

- 1) использование технологии динамического обращения к серверу «на лету», без перезагрузки всей страницы полностью, например:
 - а) с использованием XMLHttpRequest (основной объект);
 - б) через динамическое создание дочерних фреймов;
- 2) через динамическое создание тега <script>;
- 3) использование DHTML для динамического изменения содержания страницы.

В качестве формата передачи данных обычно используются JSON или XML. Впервые термин AJAX был публично использован 18 февраля 2005 года в статье Джесси Джеймса Гарретта «Новый подход к веб-приложениям». Гарретт придумал термин, когда ему пришлось как-то назвать новый набор технологий, предлагаемый им клиенту.

Однако в той или иной форме многие технологии были доступны и использовались гораздо раньше, например в подходе «Remote Scripting», предложенным компанией Microsoft в 1998 году, или с использованием HTML элемента IFRAME, появившегося в Internet Explorer 3 в 1996 году.

AJAX стал особенно популярен после использования его компанией Google в сервисах Gmail, Google Maps и Google Suggest.

Исходная инфраструктура – простая, универсальная и эффективная – стала определяющим фактором стремительного успеха модели веб-приложений. Следующее поколение веб-приложений по-прежнему будет базироваться на HTTP и страницах, однако содержимое страниц и возможности оборудования, работающего на стороне сервера, существенно изменятся. В результате впечатление от работы с веб-приложениями заметно изменится – последние приблизятся к классическим Windows-приложениям для настольных систем.

Работа сегодняшних веб-приложений основана на отправке форм, заполненных пользователем, на веб-сервер и последующем отображении разметки, возвращенной сервером. При обмене данными между браузером и сервером используется классический протокол HTTP. Как известно, HTTP относится к числу протоколов без состояния; иначе говоря, каждый запрос никак не связан с предыдущим, а автоматическое сохранение информации состояния отсутствует (объекты состояния, известные всем нам, например, по ASP.NET, представляют собой абстракцию, реализуемую средой серверного программирования).

Обмен данными между браузером и веб-сервером происходит при помощи форм. С точки зрения пользователя пересылка осуществляется постранично. Каждое действие пользователя, в результате которого серверу отправляется новый запрос, приводит к пересылке и отображению совершенно новой страницы (или измененной версии текущей страницы).

Небольшой анализ этой модели поможет выявить ее недостатки и те причины, по которым сегодня стала необходима новая модель.

На основании URL, введенного в строке адреса, браузер отображает страницу. Страница, в конечном счете, состоит из разметки HTML и содержит одну или несколько форм HTML. Пользователь вводит данные, а затем приказывает браузеру отправить форму по URL, заданному для этой цели (URL действия).

Браузер преобразует заданный URL в IP-адрес и открывает сокет. Пакет HTTP, содержащий форму вместе со всеми полями, пересылается по каналу связи заданному получателю. Веб-сервер принимает запрос и обычно передает его внутреннему модулю для дальнейшей обработки. В конце процесса создается пакет ответа HTTP, а возвращаемое браузером значение вставляется в тело пакета.

Получив запрос, например, на ресурс .aspx, веб-сервер передает его подсистеме ASP.NET для обработки и получает разметку HTML. Сгенерированная разметка включает все теги классической страницы HTML (<html>, <body>, <form> и т. д.). Исходный код страницы встраивается в ответ

HTTP и помечается соответствующим типом MIME, чтобы браузер знал, как его следует обработать. В зависимости от типа MIME браузер выбирает дальнейшие действия.

Если ответ содержит страницу HTML, браузер полностью заменяет текущее содержимое новой разметкой. Во время обработки запроса сервером на экране отображается «старая» страница. Как только «новая» страница будет полностью загружена, браузер стирает изображение и выводит ее.

В результате страницы становятся тяжелыми и громоздкими—даже если упорно говорить о «расширенной функциональности», от этого ничего не изменится.

При действующей архитектуре веб-приложений каждое действие пользователя требует полной перерисовки страницы. Как следствие, более тяжелые («широкофункциональные») страницы медленнее воспроизводятся на экране, а их прорисовка сопровождается мерцанием. При большом наборе страниц крупного приложения (например, портала) такой механизм лишь вызовет раздражение у конечного пользователя.

Хотя для построения страницы на сервере разработчик может воспользоваться целым рядом гибких архитектур с точки зрения клиента веб-страницы изначально проектировались в расчете на статичность и отсутствие изменений. В конце 1990-х годов этот основополагающий факт изменился: сначала появился стандарт Dynamic HTML, затем модель объекта документа W3C (World Wide Web Consortium). В наши дни страница, отображаемая браузером, может модифицироваться с учетом изменений, вносимых исключительно на стороне клиента, в зависимости от действий пользователя.

Стандарт Dynamic HTML стал серьезным шагом вперед, но одного его было недостаточно для качественного изменения Web.

Перерисовку страниц было необходимо свести к минимуму. Для этой цели около 1997 года появились первые примитивные формы сценарного удаленного вызова процедур (RPC, Remote Procedure Call). В частности, компания Microsoft со своей технологией RS (Remote Scripting) стала одним из новаторов в этой области.

В RS, для получения данных с удаленного URL, были задействованы апплеты Java. URL предоставлял формализованный программный интерфейс и сериализацию данных, пересылаемых в обе стороны в строковом формате. На стороне клиента компактная инфраструктура JavaScript принимала данные и активизировала функцию обратного вызова, определяемую пользователем, для обновления пользовательского интерфейса средствами Dynamic HTML или аналогичными методами. Технология RS работала в Microsoft Internet Explorer 4.0, Netscape Navigator 4.0 и последующих версиях.

Позднее компания Microsoft заменила апплет Java объектом COM с именем XMLHttpRequest и сняла большинство ограничений программного интерфейса, предоставляемого удаленным URL.

В то же время стараниями сообщества появился ряд аналогичных прикладных сред, которые должны были поднять технологию RS на новый уровень и расширить область ее практического применения. Апплет Java исчез и был заменен объектом XMLHttpRequest.

В AJAX-приложениях сочетаются два противоречивых фактора. С одной стороны, приложение должно передавать пользователям свежие данные, полученные с сервера. С другой стороны, новые данные должны интегрироваться в существующую страницу без ее полного обновления.

Основным фактором, заложенным в основу удаленного исполнения сценариев, является возможность выдачи внеполосных запросов HTTP. В данном контексте под внеполосным вызовом понимается запрос HTTP, который выдается за пределами встроенного модуля, обеспечивающего отправку форм HTTP. Внеполосный вызов инициируется событием страницы HTML и обслуживается компонентом-посредником (proxy component). В новейших AJAX-решениях таким посредником является объект XMLHttpRequest; в самых первых реализациях RS им был апплет Java.

Компонент-посредник (например, объект XMLHttpRequest) отправляет обычный запрос HTTP и дожидается (синхронно или асинхронно) завершения его обработки. Получив готовые данные ответа, посредник вызывает функцию обратного вызова JavaScript; эта функция должна обновить все части страницы, нуждающиеся в обновлении.

3.6 SMILES

SMILES (Simplified Molecular Input Line Entry System) – система правил (спецификация) однозначного описания состава и структуры молекулы химического вещества с использованием строки символов ASCII.

Представление SMILES представляет молекулярную структуру как двумерное изображение. Двумерный рисунок отдельной химической структуры возможен во многих различных формах. То есть отдельную структуру можно изобразить правильно многими различными рисунками. Таким же образом отдельную структуру можно изобразить правильно многими различными представлениями SMILES. Фактически, любая достаточно большая структура имеет множество представлений SMILES, которые правильно изображают эту структуру. Любое из правильных описаний приемлемо для компьютерной обработки. Представления SMILES составлены из атомов (обозначенных атомными символами), связей, круглых скобок (используемые для того, чтобы показать ответвления) и чисел

(используемые для того, чтобы определять позиции открытия и закрытия кольца). За исключением обозначения позиций кольца, числа не используются в представлениях SMILES. Атомы представлены своими атомными символами.

Важны символы больших и прописных букв. Все алифатические атомы введены большими буквами. Все ароматические атомы введены прописными буквами. Возможные ароматические атомы – это углерод, кислород, сера, кремний и азот. Другие потенциальные ароматические атомы в настоящее время не доступны программам SRC, потому что текущие методы оценки, используемые в программах не могут оценивать их. Атомам с двумя буквами атомного символа, такие как хлор или бром, нужно ввести первый символ большой буквой (рисунок 3.2). В случае хлора или брома, вторая буква атомного символа может быть или большой или прописной. «R» в символе брома обычно вводится строчной буквой. Предложено, чтобы буква «l» в символе хлора вводилась большой буквой («L») потому что возможно неправильное идентифицирование строчной буквы "l" и цифры один «1». Поэтому хлор можно ввести как Cl или CL и бром можно ввести как Br или BR. Существует четыре основные связи в представлении SMILES: одинарные, двойные, тройные и ароматические связи. Одинарные связи не обязательно показывать и они обычно опускаются. Одинарную связь можно обозначить символом дефиса «-». Например, правильное представление SMILES для пропана: C-C-C; однако, нет преимуществ ввода одинарной связи. Поэтому она обычно не используется (программы SRC автоматически удаляют любые дефисы, введенные в строку SMILES)

В изначальной спецификации SMILES отсутствуют правила, касающиеся способа построения записи и способа различения пространственных изомеров молекул. Для решения этих проблем были разработаны расширения стандарта.

«Каноническая SMILES» (Canonical SMILES) – версия спецификации, включающая правила канонизации, позволяющие записать формулу молекулы любого вещества однозначным образом. Эти правила касаются выбора первого атома в записи, направления обхода циклов, выбора направления основной цепи при разветвлениях. Поскольку в разных пакетах молекулярного моделирования используются различные алгоритмы канонизации SMILES, вследствие чего могут получаться разные записи одной и той же молекулы, понятие «каноническая SMILES» не является абсолютным. Данная версия стандарта обычно применяется для индексирования и проверки уникальности молекул в базах данных.

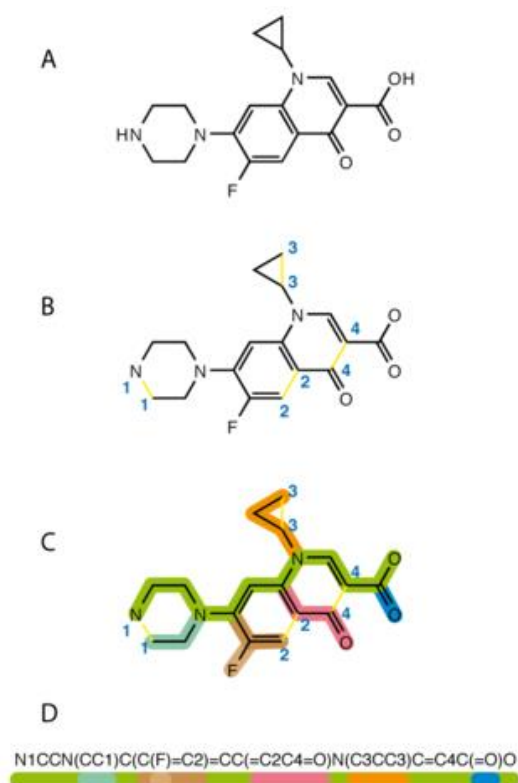


Рисунок 3.2 – Генерация SMILES: сначала кодировке подвергаются разорванные кольца, затем описываются ответвления от основной структуры

В терминах теории графов SMILES представляет собой строку, полученную путём вывода символов вершин молекулярного графа в порядке, соответствующем их обходу в глубину. Первоначальная обработка графа включает в себя удаление атомов водорода и разбивку циклов таким образом, чтобы получившийся граф представлял собой остовный лес. Местам разбиения графа ставятся в соответствие числа, показывающие наличие связи в исходной молекуле. Для указания точек ветвления молекулы используются скобки.

Двойная связь обозначена символом равенства «=» и необходимо идентифицировать двойную связь. Следующие примеры иллюстрируют двойную связь (таблица 3.1):

Таблица 3.1 –Примеры двойных связей SMILES

Состав	Молекулярная формула	Представление SMILES
Этилен	CH ₂ =CH ₂	C=C
Пропилен	CH ₂ =CH-CH ₃	C=CC
2-Бутен	CH ₃ -CH=CH-CH ₃	CC=CC

Ветви в молекулярных структурах обозначены вложениями в круглых скобках. Когда структура содержит ветвь, представление SMILES структуры требует, чтобы ветвь была обозначена вложениями в круглых скобках (рисунок 3.3).

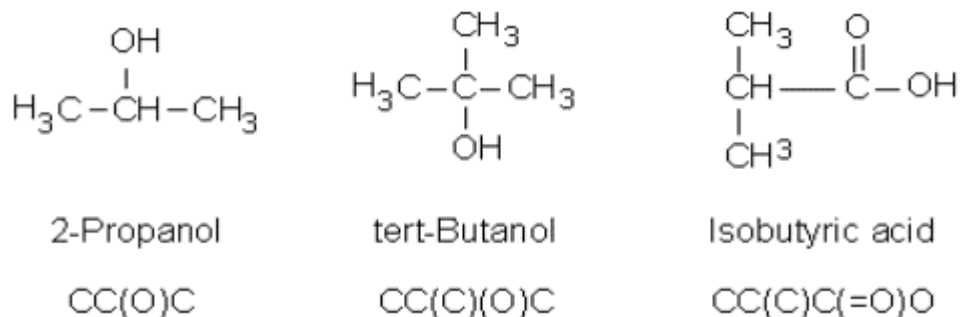


Рисунок 3.3 – Примеры представления ответвлений

Наиболее трудным аспектом написания представления SMILES является написание правильного представления SMILES для сложной кольцевой системы. Однако, написание представления SMILES для структур, содержащих только одно или два кольца, довольно просто. Следующие правила кодирования применяются ко всем циклическим структурам:

- 1) циклическим структурам требуются числа, чтобы указать, где начинается и заканчивается кольцо. Числа от 1 до 9 используются, чтобы указать начальные и конечные атомы;
- 2) тот же самый номер используется, чтобы указать начальный и конечный атом для каждого кольца. Начальный и конечный атом должны быть связаны друг с другом;
- 3) каждый используемый номер (1, 2, 3 и т.д.) должен появиться дважды и только дважды в полном представлении SMILES;
- 4) числа вводятся сразу же после атомов для того, чтобы указывать начальные и конечные позиции. Например, номер не может следовать за ветвью;
- 5) начальный или конечный атом могут быть связаны с двумя последовательными числами [9].

3.7 Model-View-Controller

Основная цель применения этой концепции состоит в отделении бизнес-логики (модели) от её визуализации (представления, вида). За счёт такого разделения повышается возможность повторного использования кода. Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных

контекстах и/или с различных точек зрения. В частности, выполняются следующие задачи. К одной модели можно присоединить несколько видов, при этом не затрагивая реализацию модели. Например, некоторые данные могут быть одновременно представлены в виде электронной таблицы, гистограммы и круговой диаграммы.

Не затрагивая реализацию видов, можно изменить реакции на действия пользователя (нажатие мышью на кнопке, ввод данных) – для этого достаточно использовать другой контроллер;

Модель предоставляет данные и методы работы с ними: запросы в базу данных, проверка на корректность. Модель не зависит от представления (не знает как данные визуализировать) и контроллера (не имеет точек взаимодействия с пользователем) просто предоставляя доступ к данным и управлению ими. Модель строится таким образом, чтобы отвечать на запросы, изменяя своё состояние, при этом может быть встроено уведомление «наблюдателей». Модель, за счёт независимости от визуального представления, может иметь несколько различных представлений для одной «модели».

Представление отвечает за получение необходимых данных из модели и отправляет их пользователю. Представление не обрабатывает введённые данные пользователя. Представление может влиять на состояние модели, сообщая модели об этом.

Контроллер обеспечивает «связи» между пользователем и системой. Контролирует и направляет данные от пользователя к системе и наоборот. Использует модель и представление для реализации необходимого действия.

Поскольку MVC не имеет строгой реализации, то реализован он может быть по-разному. Нет общепринятого определения, где должна располагаться бизнес-логика. Она может находиться как в контроллере, так и в модели. В последнем случае, модель будет содержать все бизнес-объекты со всеми данными и функциями.

Некоторые фреймворки жестко задают где должна располагаться бизнес-логика, другие не имеют таких правил.

Интернационализация и форматирование данных также не имеет четких указаний. Для реализации схемы «Model-View-Controller» используется достаточно большое число шаблонов проектирования (в зависимости от сложности архитектурного решения), основные из которых – «наблюдатель», «стратегия», «компоновщик».

Наиболее типичная реализация – в которой представление отделено от модели путём установления между ними протокола взаимодействия, использующего «аппарат событий» (обозначение «событиями» определённых ситуаций, возникающих в ходе выполнения программы, – и рассылка уведомлений о них всем тем, кто подписался на получение): при каждом

особом изменении внутренних данных в модели (обозначенном как «событие»), она оповещает о нём те зависящие от неё представления, которые подписаны на получение такого оповещения – и представление обновляется. Так используется шаблон «наблюдатель».

При обработке реакции пользователя – представление выбирает, в зависимости от реакции, нужный контроллер, который обеспечит ту или иную связь с моделью. Для этого используется шаблон «стратегия», или вместо этого может быть модификация с использованием шаблона «команда».

Для возможности однотипного обращения с подобъектами сложно-составного иерархического вида – может использоваться шаблон «компоновщик». Кроме того, могут использоваться и другие шаблоны проектирования – например, «фабричный метод», который позволит задать по умолчанию тип контроллера для соответствующего вида. по расположению.

4 ОПИСАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ

4.1 CountVectorizer

Одна из первых концепций обработки языков Bag-of-Words – это статистический анализ, анализирующий количественное вхождение слов в документах. Он отлично подходит для обработки текстов и прототипирования. Алгоритм CountVectorizer как раз позволяет сконвертировать набор текстов в матрицу сигналов, находящихся в тексте, для нахождения различий одних данных от других. Также можно задать минимальное количество необходимое для появления токена в матрице и даже получить статистику по n-граммам. Следует учитывать, что CountVectorizer по умолчанию производит токенизацию. Необходимо понимать, что этот алгоритм работает только на строках больше самых минимальных.

Из недостатков этого метода можно выделить плохую масштабируемость на большой набор текстов. Алгоритм требует большое количество ресурсов для работы с большим (>10000) количеством документов. Поэтому надо либо ограничивать количество токенов для попадания в словарь или же использовать другой, более оптимальный алгоритм. Однако для нескольких тысяч можно весьма быстро получить результат и далее использовать различия например для классификации или кластеризации документов, что является целью.

CountVectorizer преобразовывает входной текст в матрицу, значениями которой, являются количества вхождения данного ключа(слова) в текст. В отличие от FeatureHasher имеет лучшую гибкость, но работает медленнее. Для более точного понимания работы CountVectorizer приведем простой пример. Допустим есть массив с текстовыми значениями: “раз два три”, “три четыре два два”, “раз раз раз четыре”.

Для начала CountVectorizer собирает уникальные ключи из всех записей, в нашем примере это будет: [раз, два, три, четыре].

Длина списка из уникальных ключей и будет длиной закодированного текста (в нашем случае это 4). А номера элементов будут соответствовать, количеству раз встречи данного ключа с данным номером в строке: раз два три [1,1,1,0], три четыре два два [0,2,1,1]. Значения после обработки: “1,1,1,0”, “0,2,1,1”, “3,0,0,1”.

Пример работы:

```
if isinstance(raw_documents, six.string_types):
    raise ValueError()
self._validate_params()
self._validate_vocabulary()
max_df = self.max_df
min_df = self.min_df
max_features = self.max_features
vocabulary, X = self._count_vocab(raw_documents,
                                   self.fixed_vocabulary_)

if self.binary:
    X.data.fill(1)
if not self.fixed_vocabulary_:
    X = self._sort_features(X, vocabulary)

n_doc = X.shape[0]
max_doc_count = (max_df
                  if isinstance(max_df, numbers.Integral)
                  else max_df * n_doc)
min_doc_count = (min_df
                  if isinstance(min_df, numbers.Integral)
                  else min_df * n_doc)
if max_doc_count < min_doc_count:
    raise ValueError()
X, self.stop_words_ = self._limit_features(X, vocabulary,
                                           max_doc_count,
                                           min_doc_count,
                                           max_features)

self.vocabulary_ = vocabulary
return X
```

Соответственно после кодировки, применения данного метода мы получим следующий массив (рисунок 4.1):

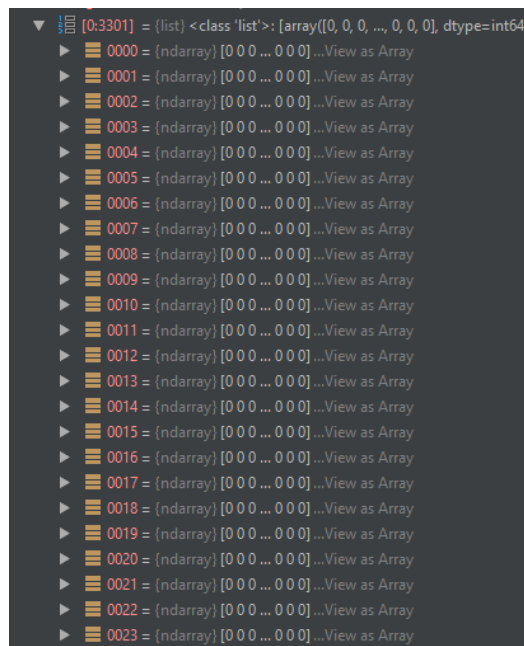


Рисунок 4.1 – Массив из найденных различий, на обучающей выборке

4.2 Дерево решений

Деревья решений используются в повседневной жизни в самых разных областях человеческой деятельности, порой и очень далеких от машинного обучения. Деревом решений можно назвать наглядную инструкцию, что делать в какой ситуации. В терминах машинного обучения можно сказать, что это элементарный классификатор, который определяет форму по нескольким признакам. Дерево решений как алгоритм машинного обучения – по сути то же самое: объединение логических правил вида "Значение признака А меньше Х И Значение признака В меньше Y следовательно Класс 1" в структуру данных "Дерево". Огромное преимущество деревьев решений в том, что они легко интерпретируемы, понятны человеку.

Листьями дерева принятия решений являются классы. Чтобы классифицировать объект при помощи дерева принятия решений – нужно последовательно спускаться по дереву (выбирая направление основываясь на значениях предикатов применяемых к классифицируемому объекту). Путь от корня дерева до листьев можно трактовать как объяснение того, почему тот или иной объект отнесен к какому-либо классу.

Также, не накладывается ограничений на значения атрибутов объекта – они могут иметь как категориальную, так и числовую или логическую природу. Нужно только определить предикаты, которые умеют правильно обрабатывать значения атрибутов (например, вряд ли есть смысл использовать

предикаты «больше» или «меньше» для атрибутов с логическими значениями).

$s0$ = вычисляем энтропию исходного множества

Если $s0 == 0$ значит:

Все объекты исходного набора, принадлежат к одному классу

Сохраняем этот класс в качестве листа дерева

Если $s0 \neq 0$ значит:

Ищем предикат, который разбивает исходное множество таким образом чтобы уменьшилось среднее значение энтропии

Найденный предикат является частью дерева принятия решений, сохраняем его

Разбиваем исходное множество на подмножества, согласно предикату

Повторяем данную процедуру рекурсивно для каждого подмножества

В основе популярных алгоритмов построения дерева решений, таких как ID3 и C4.5, лежит принцип жадной максимизации прироста информации – на каждом шаге выбирается тот признак, при разделении по которому прирост информации оказывается наибольшим. Далее процедура повторяется рекурсивно, пока энтропия не окажется равной нулю или какой-то малой величине (если дерево не подгоняется идеально под обучающую выборку во избежание переобучения).

Главным достоинством является, получаемая в результате, древовидная структура предикатов, которая позволяет интерпретировать результаты классификации (хотя в силу своей «жадности», описанный алгоритм, не всегда позволяет обеспечить оптимальность дерева в целом).

В разных алгоритмах применяются разные эвристики для "ранней остановки" или "отсечения", чтобы избежать построения переобученного дерева [10].

5 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ

5.1 Введение и исходные данные

Применение данного программного средства позволит упростить разработку лекарственных препаратов. Используя встроенные инструменты моделирования программный продукт позволяет в короткие сроки получить результат. Эти данные рекомендуется учитывать перед началом пре-клинических испытаний лекарственных средств. Кроме того, система позволит найти ошибки в медикаментозном и в побочном эффекте после применения.

Целью технико-экономического обоснования программного средства является определение экономической выгоды создания данного программного продукта и дальнейшего применения. В данном разделе необходимо вычислить стоимость разрабатываемого программного продукта, определить экономическую выгоду его создания и дальнейшего применения.

Программный комплекс относится к 1-й группе сложности. Для оценки экономической эффективности разработанного программного средства проводится расчет прибыли от продажи одной системы (программы). Расчеты выполнены на основе методического пособия.

5.2 Расчет сметы затрат и цены программного продукта

В таблице 5.1 приведены исходные данные для расчета сметы затрат и цены программного продукта.

Таблица 5.1 – Исходные данные для расчета сметы затрат и цены программного продукта

Наименование показателей	Буквенные обозначения	Единицы измерения	Количество
Коэффициент новизны	K_n	единиц	1,0

Продолжение таблицы 5.1

Наименование показателей	Буквенные обозначения	Единицы измерения	Количество
Группа сложности		единиц	1
Дополнительный коэффициент сложности	$K_{сл}$	единиц	0,07
Поправочный коэффициент, учитывающий использование типовых программ	K_T	единиц	0,6
Установленная плановая продолжительность разработки	T_p	лет	0,8
Продолжительность рабочего дня	$T_ч$	ч	8
Тарифная ставка 1-го разряда	$T_{м1}$	руб.	33
Коэффициент премирования	$K_{п}$	единиц	0,6
Норматив дополнительной заработной платы исполнителей	H_d	%	20
Отчисления в фонд социальной защиты населения	$З_{сз}$	%	34
Отчисления в Белгосстрах	$K_{нс}$	%	0,6
Расходы на научные командировки	$P_{нкi}$	%	30
Прочие прямые расходы	$P_{зи}$	%	20

Объем программного средства определяется путем подбора аналогов на основании классификации типов программного средства, каталога функций, которые постоянно обновляются и утверждаются в установленном порядке. На основании информации и функциях разрабатываемого программного

средства по каталогу функций определяется объем функций. Объем программного средства определяется на основе нормативных данных, приведенных в таблице 5.2.

Таблица 5.2 – Характеристика функций и их объем

Номер функции	Наименование (содержание) функций	Объем функций
101	Организация ввода информации	400
102	Контроль, предварительная обработка и ввод информации	300
109	Организация ввода/вывода информации в интерактивном режиме	770
111	Управление вводом/выводом	240
207	Манипулирование данными	350
302	Сортировка файла	800
305	Обработка файлов	350
308	Управление файлами	3690
309	Формирование файла	1250
507	Обеспечение интерфейса между компонентами	1750
701	Математическая статистика и прогнозирование	4680
703	Расчет показателей	610
705	Формирование и вывод на внешние носители	1840
Итого:		17030

Общий объем программного продукта (V_0) определяется по следующей формуле (5.1):

$$V_0 = \sum_{i=1}^n V_i, \quad (5.1)$$

где V_i – объем отдельной функции ПО;
 n – общее число функций.

$$V_0 = 17030 \text{ (строк исходного кода).}$$

Исходя из режима работы в реальном времени, а также обеспечения существенного распараллеливания вычислений, применяется коэффициент K_c к объему ПО, который определяется по формуле (5.2):

$$K_c = 1 + \sum_{i=1}^n K_i, \quad (5.2)$$

где K_i – коэффициент, соответствующий степени повышения сложности за счет конкретной характеристики;
 n – количество учитываемых характеристик.

$$K_c = 1 + 0,07 = 1,07.$$

С учетом дополнительного коэффициента сложности K_c рассчитывается общая трудоемкость ПС по формуле (5.3):

$$T_o = T_n \cdot K_c, \quad (5.3)$$

где T_o – общая трудоемкость;
 T_n – нормативная трудоемкость ПС;
 K_c – дополнительный коэффициент сложности ПС;

$$T_o = 520 * 1,07 = 556.4 \text{ чел./дн.}$$

На основе уточненной трудоемкости разработки ПС и установленного периода разработки, общая плановая численность разработчиком рассчитывается по формуле (5.4):

$$\chi_p = \frac{T_o}{T_p * \Phi_{эф}}, \quad (5.4)$$

где $\Phi_{эф}$ – эффективный фонд времени работы одного работника в течение года (дн.);
 T_o – общая трудоемкость разработки проекта (чел./дн.);
 T_p – срок разработки проекта (лет).

Эффективный фонд времени работы одного работника ($\Phi_{эф}$) рассчитывается по формуле (5.5):

$$\Phi_{эф} = D_r - D_{п} - D_{в} - D_o, \quad (5.5)$$

где D_r – количество дней в году;
 $D_{п}$ – количество праздничных дней в году;
 $D_{в}$ – количество выходных дней в году;
 D_o – количество дней отпуска.

$$\Phi_{эф} = 230 \text{ дней в году.}$$

Срок разработки установлен 14 месяцев ($T_p = 1,2 \text{ г.}$):

$$Ч_p = \frac{556.4}{0,8 \cdot 230} \approx 3,02 \approx 3 \text{ человек.}$$

Рассчитаем трудоемкость ПС и численность исполнителей по стадиям по таблице 5.3.

Таблица 5.3 – Расчет уточненной трудоемкости ПС и численности исполнителей по стадиям

Показатели	Стадии					Итого
	ТЗ	ЭП	ТП	РП	ВН	
Коэффициенты удельных весов трудоемкости стадии разработки ПО (d)	0,11	0,09	0,11	0,55	0,14	1,0
Коэффициент сложности ПО (K_c)	1,07	1,07	1,07	1,07	1,07	
Коэффициент, учитывающий использование стандартных модуле				0,6		

Продолжение таблицы 5.3

Показатели	Стадии					Итого
Коэффициент, учитывающий новизну ПО (K_n)	1,0	1,0	1,0	1,0	1,0	
Численность исполнителей, чел. ($Ч_i$)	3	3	3	3	3	3
Сроки разработки, лет						0,8

Реализацией проекта занимались 2 человека. В соответствии с численностью и выполняемым функциями устанавливается штатное расписание группы специалистов-разработчиков.

Расчет основной заработной платы осуществляется в следующей последовательности. Определим месячные (T_m) и часовые ($T_{\text{ч}}$) тарифные ставки начальника отдела (тарифный разряд – 14; тарифный коэффициент – 5,31;), инженера-программиста 1-й категории (тарифный разряд – 12; тарифный коэффициент – 4,39). Месячный тарифный оклад ($T_{\text{мо}}$) определяется путем умножения действующей месячной тарифной ставки 1-го разряда ($T_{\text{м1}}$) на тарифный коэффициент ($T_{\text{ки}}$), соответствующий установленному тарифному разряду специалиста (формула (5.6)):

$$T_m = T_{\text{м1}} \cdot T_{\text{к}}. \quad (5.6)$$

Часовая тарифная ставка рассчитывается путем деления месячной тарифной ставки на установленный при сорокачасовой рабочей неделе в восьмичасовом рабочем дне фонд рабочего времени – 168 часов – приведена в формуле (4.7)

$$T_{\text{ч}} = \frac{T_m}{168}, \quad (5.7)$$

где $T_{\text{ч}}$ – часовая тарифная ставка (руб.);
 T_m – месячная тарифная ставка (руб.).

Принимаем тарифную ставку 1-го разряда равной 33 руб. Месячная и часовая тарифные ставки начальника отдела:

$$T_M = 33 * 5,31 = 175,23 \text{ руб.},$$

$$T_ч = \frac{175,23}{168} = 1,04 \text{ руб.}$$

Месячная и часовая тарифные ставки инженера-программиста 1-й категории равны соответственно:

$$T_M = 33 * 4,39 = 144,87 \text{ руб.},$$

$$T_ч = \frac{144,87}{168} = 0,86 \text{ руб.}$$

Расчет месячных и почасовых тарифных ставок сведен в таблицу 5.4.

Таблица 5.4 – Штатное расписание группы разработчиков

Должность	Количество ставок	Тарифный разряд	Тарифный коэффициент	Месячная тарифная ставка (руб.)	Часовая тарифная ставка (руб.)
Начальник отдела (ведущий инженер программист)	1,00	14	5,31	175,23	1,04
Инженер-программист 1-й категории	1,00	12	4,39	144,87	0,86
Итого:				320,1	1,9

Основная заработная плата исполнителей на конкретное ПО рассчитывается по формуле (4.8):

$$Z_o = \sum_{i=1}^n T_ч^i \cdot T_ч \cdot \Phi_{\Pi} \cdot K, \quad (5.8)$$

где n – количество исполнителей, занятых разработкой конкретного ПО;
 $T_ч^i$ – часовая тарифная ставка i -го исполнителя;
 Φ_{Π} – плановый фонд рабочего времени i -го исполнителя;
 $T_ч$ – количество часов работы в день;
 K – коэффициент премирования;

$$З_0 = 1,04 * 8 * 152,5 * 0,6 + 4 * 0,86 * 8 * 152,5 * 0,6 = 3279 \text{ руб.}$$

Дополнительная заработная плата исполнителей проекта определяется по формуле (4.9):

$$З_д = \frac{З_0 * Н_д}{100} = \frac{3279 * 20}{100} = 655,8 \text{ руб.} \quad (5.9)$$

Отчисления в фонд социальной защиты населения и на обязательное страхование ($З_{сз}$) определяются в соответствии с действующими законодательными актами по формуле (4.10):

$$З_{сз} = \frac{(З_0 + З_д) * Н_{сз}}{100} = \frac{(3279 + 655,8) * 34,6}{100} = 1361,44 \text{ руб.} \quad (5.10)$$

где $Н_{сз}$ – норматив отчислений в фонд социальной защиты населения и на обязательное страхование (34 +0,6%).

Расходы по статье «Машинное время» (P_M) включают оплату машинного времени, необходимого для разработки и отладки ПС. Они определяются в машино-часах по нормативам на 100 строк исходного кода машинного времени в зависимости от характера решаемых задач и типа ПО, и определяются по формуле (4.11):

$$P_M = Ц_m * \frac{V_0}{100} * T_p = 1 * \frac{17030}{100} * 0,8 = 136,24 \text{ руб.} \quad (5.11)$$

Затраты по статье «Накладные расходы» (P_H), связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных (экспериментальных) производств, а также с расходами на общехозяйственные нужды (P_H), определяются по формуле (5.12):

$$P_H = \frac{З_0 * Н_{pH}}{100} = \frac{3279 * 100}{100} = 3279 \text{ руб.} \quad (5.12)$$

Полная себестоимость высчитывается по формуле (5.13):

$$C_{п} = З_0 + З_д + P_M + З_{сз} + P_H = 8711,48 \text{ руб.} \quad (5.13)$$

Прогнозируемая прибыль $\Pi_{\text{пс}}$ рассчитывается по формуле (5.14):

$$\Pi_{\text{пс}} = \frac{C_{\text{п}} * Y_{\text{р}}}{100} = \frac{8711,48 * 40}{100} = 3484,59 \text{ руб.} \quad (5.14)$$

Прогнозируемая отпускная цена $\Pi_{\text{п}}$ вычисляется по формуле (5.15):

$$\Pi_{\text{п}} = C_{\text{п}} + \Pi_{\text{пс}} = 8711,48 + 3484,59 = 12196,07 \text{ руб.} \quad (5.15)$$

Налог на добавленную стоимость (НДС_и) считается по формуле (5.16):

$$\text{НДС} = \frac{\Pi_{\text{п}} * H_{\text{дс}}}{100} = 2439,21 \text{ руб,} \quad (5.16)$$

где $H_{\text{дс}}$ – норматив НДС (%).

Для прогнозируемой отпускной цены ($\Pi_{\text{о}}$) существует формула (5.17):

$$\Pi_{\text{о}} = \Pi_{\text{п}} + \text{НДС} = 12196,07 + 2439,21 = 14635,28 \text{ руб.} \quad (5.17)$$

Кроме того, организация-разработчик осуществляет затраты на сопровождение $P_{\text{с}}$, которые определяются по нормативу ($H_{\text{с}}$), где $H_{\text{с}}$ – норматив расходов на сопровождение и адаптацию (20%) – формула (5.18).

$$P_{\text{с}} = \frac{C_{\text{п}} * H_{\text{с}}}{100} = 1742,29 \text{ руб.} \quad (5.18)$$

В следующей таблице 5.5 приведены исходные данные для расчета экономического эффекта.

Таблица 5.5 – Исходные данные для расчета экономического эффекта

Наименование показателей	Обозначения	Единицы измерения	Значение		Наименование источника информации
			в базовом варианте	в новом варианте	
Капитальные вложения, включая затраты пользователя на приобретение	$K_{пр}$	руб.		14635,28	Договор заказчика с разработчиком
Затраты на сопровождение ПО	K_c	руб.		11876	Договор заказчика с разработчиком
Время простоя сервиса, обусловленное ПО, в день	P_1, P_2	мин	200	20	Расчетные данные пользователя и паспорт ПО
Стоимость одного часа простоя	C_p	руб.	100	100	Расчетные данные пользователя и паспорт ПО
Среднемесячная ЗП одного программиста	$З_{см}$	руб.	144,87	144,87	Расчетные данные пользователя
Коэффициент начислений на зарплату	$K_{нз}$		0,6	0,6	Рассчитывается по данным пользователя
Среднемесячное количество рабочих дней	D_p	день	21	21	Принято для расчета

Продолжение таблицы 5.5

Наименование показателей	Обозначения	Единицы измерения	Значение		Наименование источника информации
			в базовом	в новом варианте	
Количество типовых задач, решаемых за год	Z_{T1}, Z_{T2}	задача	1500	1500	План пользователя
Объем выполняемых работ	A_1, A_2	задача	1000	1000	План пользователя
Средняя трудоемкость работ на задачу	T_{c1}, T_{c2}	Человеко-часов	15	1.3	Рассчитывается по данным пользователя
Количество часов работы в день	$T_{\text{ч}}$	ч	8	8	Принято для расчета
Ставка налога на прибыль	$H_{\text{п}}$	%		18	

5.3 Расчет капитальных затрат

Экономия затрат на заработную плату в расчете на 1 задачу ($C_{зе}$) – формула (5.19):

$$C_{зе} = \frac{Z_{\text{см}} * (T_{c1} - T_{c2})}{D_{\text{р}} * T_{\text{ч}}} = 11,81 \text{ руб}, \quad (5.19)$$

где $Z_{\text{см}}$ – среднемесячная заработная плата одного программиста (руб.);
 T_{c1}, T_{c2} – снижение трудоемкости работ в расчете на 1 задачу (человеко-часов);
 $T_{\text{ч}}$ – количество часов работы в день (ч);
 $D_{\text{р}}$ – среднемесячное количество рабочих дней.

Экономия заработной платы при использовании нового ПО (тыс. руб.) – формула (5.20):

$$C_3 = C_{зе} * A_2 = 11.81 * 1000 = 118.10 \text{ руб}, \quad (5.20)$$

где $C_{зе}$ – экономия заработной платы типовой задачи;
 A_2 – количество типовых задач, решаемых за год (задач).

Экономия с учетом начисления на зарплату (C_n) высчитывается по формуле (5.21):

$$C_n = C_z * K_{нз} = 118.100 * 0,6 = 7.086 \text{ руб.} \quad (5.21)$$

Экономия за счет сокращения простоев сервиса (C_c) рассчитывается по формуле (4.22):

$$C_c = \frac{(П_1 - П_2) * D_{рг} * C_{п}}{60} = 6.300 \text{ руб.} \quad (5.22)$$

где $D_{рг}$ – плановый фонд работы сервиса (дней).

Общая готовая экономия текущих затрат, связанных с использованием нового ПО (C_0), рассчитывается по формуле (5.23):

$$C_0 = C_n + C_c = 13386 \text{ руб.} \quad (5.23)$$

5.4 Расчет экономического эффекта

Внедрение нового ПО позволит пользователю сэкономить на текущих затратах, т.е. практически получить на эту сумму дополнительную прибыль. Для пользователя в качестве экономического эффекта выступает лишь чистая прибыль – дополнительная прибыль, остающаяся в его распоряжении ($\Delta П_ч$), которая определяется по формуле (5.24):

$$\Delta П_ч = C_0 - \frac{C_0 * H_{п}}{100} = 10.976.52 \text{ руб.} \quad (5.24)$$

В процессе использования нового ПО чистая прибыль в конечном итоге возмещает капитальные затраты. Однако полученные при этом суммы результатов (прибыли) и затрат (капитальных вложений) по годам приводят к единому времени – расчетному году (за расчетный год принят 2018-й год)

путем умножения результатов и затрат за каждый год на коэффициент дисконтирования α . В данном примере используются коэффициенты: 2018 г. – 1, 2019-й – 0,8696, 2020-й – 0,7561, 2021 г. – 0,6575. Все рассчитанные данные экономического эффекта сводятся в таблицу 5.6 [11].

Таблица 5.6 – Расчет экономического эффекта от использования нового программного средства

Показатели	Единицы измерения	Годы			
		2018	2019	2020	2021
Результаты					
Прирост прибыли за счет экономии затрат (Пч)	руб.		10.976.52	10.976.52	10.976.52
То же с учетом фактора времени	руб.		9545.18	8299.34	7217.06
Приобретение ПО (Кпр)	руб.	14635, 28			
Сопровождение (Кс)	руб.	1742, 29			
Всего затрат	руб.	16377.57			
То же с учетом фактора времени	руб.	16377.57			
Экономический эффект					
Превышение результата над затратами	руб.	-16377,57	9545,18	8299,34	7217,06
То же с нарастающим итогом	руб.	-16377,57	-6831,82	1467,52	8684,58
Коэффициент приведения	единицы	1	0,8696	0,7561	0,6575

5.5 Вывод по технико-экономическому обоснованию

Исходя из таблицы расчета экономического эффекта использования рекомендательной системы оценки терапевтической активности веществ можно прийти к выводу, что система является экономически выгодной и ориентирована на пользователей, имеющих специальную подготовку. Положительный экономический эффект заключается в значительном уменьшении трудоемкости работ на задачу. Продукт является экономически выгодным благодаря тому, что его окупаемость достигается по истечении двух лет. К концу этого срока чистая прибыль составит 8684,58 рублей.

ЗАКЛЮЧЕНИЕ

В ходе работы над дипломным проектом была сформулирована и поставлена задача по разработке фармакологической системы оценки терапевтической активности веществ, которая позволит в большей степени оптимизировать процесс подбора комбинаторной библиотеки, а также упростить молекулярный докинг при разработке оригинального лекарственного средства.

Фармакологическая система была разработана с целью повышения производительности и качества процесса драг-дизайна, что позволит сделать его более эффективным.

Для того чтобы правильно спроектировать систему, необходимо было грамотно проработать предметную область, также разработать основной процесс предметной области. Это позволило подробно разобраться и понять, что должно выполнять приложение.

Важным этапом явилось проектирование такого интерфейса системы, чтобы подготовленному пользователю было привычно пользоваться приложением.

Система может быть улучшена за счет добавления алгоритмов докига. Такой функционал уместен при профессиональной и коммерческой разработке препаратов. Такой функционал может быть реализован в виде мобильного приложения, синхронизирующийся с основной шиной данных.

Согласно проведенному технико-экономическому обоснованию разрабатываемая система поддержки работы департамента взыскания является экономически эффективной. По расчетам выявлено, что все дополнительные капитальные затраты на освоение, сопровождение и адаптацию нового ПО окупятся в течение второго года эксплуатации. Положительный экономический эффект достигнут за счет экономии затрат на заработную плату.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

- [1] БИОМОЛЕКУЛА [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://biomolecula.ru/> – Дата доступа: 05.04.18.
- [2] Яранцева Н.Д. – Современная методология создания оригинальных лекарственных средств.
- [3] FDA [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.fda.gov/Drugs/default.htm> – Дата доступа: 03.04.18.
- [4] CHEMNET: Химические наука и образование в России [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://chem.msu.su/rus/journals/xr/chel.html> – Дата доступа: 07.04.18.
- [5] ПРОВИЗОР [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://provisor.com.ua/archive/2007/N16/develop.php> – Дата доступа: 01.04.18.
- [6] Национальное общество доказательно фармакологии [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://cardiodrug.ru/pages/terapevticheskaya-ekvivalentnost-1.html> – Дата доступа: 15.04.18.
- [7] Python: 3.6.5 documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.python.org/3/> – Дата доступа: 11.04.18.
- [8] Django documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.djangoproject.com/en/2.0/> – Дата доступа: 09.04.18.
- [9] Daylight: Chemical Information Systems [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://www.daylight.com/dayhtml/doc/theory/theory.smiles.html> – Дата доступа: 05.04.18.
- [10] И. Куралёнок, Н. Поваров – Деревья решений.
- [11] Палицын, В.А. Техничко-экономическое обоснование дипломных проектов: Метод. Пособие для студ. всех спец. БГУИР. В 4-х ч. Ч.4: Проекты программного обеспечения / В.А. Палицын. – Минск: БГУИР, 2006. – 76с

ПРИЛОЖЕНИЕ А

(обязательное)

Код классов и алгоритмов системы

```
function getPrediction() {  
    var smilesInput = $("#jme_output").val();  
    $("#therapyName").text("Please wait...");  
    $.ajax({  
        url: '/predict',  
        data: {  
            'smiles': smilesInput  
        },  
        dataType: 'json',  
        success: function (data) {  
            $("#therapyName").text(data.therapy);  
        }  
    });  
};
```

```
from django.urls import path
```

```
from . import views
```

```
urlpatterns = [  
    path("", views.index, name='index'),  
    path('predict', views.predict, name='predict'),  
]
```

```
from django.shortcuts import render  
from django.http import JsonResponse  
import web.BL.predict
```

```
def index(request):  
    return render(request, 'web/index.html', { })
```

```
def predict(request):
```

```

data = {"therapy" : "Please generate SMILES"}
if request.GET.get('smiles', None) != str():
    output = web.BL.predict.predict(request.GET.get('smiles', None))
    data["therapy"] = output

return JsonResponse(data)

def predict(inputPattern):
    train_data_set = 3300
    test_data_set = 3301
    all_data_set = 3300

    data = pd.read_csv("G:/Labs/bsuir-
neural/neural/web/BL/SELECT_DISTINCT____d_DrugKey____dr_cd_s_t
herapy_notnull.csv").as_matrix()
    clf = DecisionTreeClassifier()
    xtrainstr = np.array([])

    for chem in data[0:all_data_set, 1]: ## data limit
        xtrainstr = np.append(xtrainstr, re.sub("\ \.*\|", "", chem))
    xtrainstr = np.append(xtrainstr, re.sub("\ \.*\|", "", inputPattern))
    alldata = []

    for arr in CountVectorizer().fit_transform(xtrainstr).A:
        alldata.append(list(arr))
    castxtrain = alldata[0:train_data_set-1]
    castxtest = [alldata[len(alldata)-1]]
    all_label = np.array([], dtype=object)
    for label in data[0:all_data_set, 5]:
        all_label = np.append(all_label, letter_dict[label])
    train_label = list(all_label)[0:train_data_set-1]
    clf.fit(castxtrain, train_label)
    p = clf.predict(castxtest)
    return therapy_dict[p[0]]

```

```

class CountVectorizer(BaseEstimator, VectorizerMixin):
    def __init__(self, input='content', encoding='utf-8',
                  decode_error='strict', strip_accents=None,
                  lowercase=True, preprocessor=None, tokenizer=None,
                  stop_words=None, token_pattern=r"(?u)\b\w\w+\b",
                  ngram_range=(1, 1), analyzer='word',
                  max_df=1.0, min_df=1, max_features=None,
                  vocabulary=None, binary=False, dtype=np.int64):
        self.input = input
        self.encoding = encoding
        self.decode_error = decode_error
        self.strip_accents = strip_accents
        self.preprocessor = preprocessor
        self.tokenizer = tokenizer
        self.analyzer = analyzer
        self.lowercase = lowercase
        self.token_pattern = token_pattern
        self.stop_words = stop_words
        self.max_df = max_df
        self.min_df = min_df
        if max_df < 0 or min_df < 0:
            raise ValueError("negative value for max_df or min_df")
        self.max_features = max_features
        if max_features is not None:
            if (not isinstance(max_features, numbers.Integral) or
                max_features <= 0):
                raise ValueError(
                    "% max_features")
        self.ngram_range = ngram_range
        self.vocabulary = vocabulary
        self.binary = binary
        self.dtype = dtype

    def _sort_features(self, X, vocabulary):
        sorted_features = sorted(six.iteritems(vocabulary))
        map_index = np.empty(len(sorted_features), dtype=np.int32)

```

```

for new_val, (term, old_val) in enumerate(sorted_features):
    vocabulary[term] = new_val
    map_index[old_val] = new_val

```

```

X.indices = map_index.take(X.indices, mode='clip')
return X

```

```

def _limit_features(self, X, vocabulary, high=None, low=None,
                    limit=None):

```

```

    if high is None and low is None and limit is None:
        return X, set()

```

```

    dfs = _document_frequency(X)
    tfs = np.asarray(X.sum(axis=0)).ravel()
    mask = np.ones(len(dfs), dtype=bool)
    if high is not None:
        mask &= dfs <= high
    if low is not None:
        mask &= dfs >= low
    if limit is not None and mask.sum() > limit:
        mask_inds = (-tfs[mask]).argsort()[:limit]
        new_mask = np.zeros(len(dfs), dtype=bool)
        new_mask[np.where(mask)[0][mask_inds]] = True
        mask = new_mask

```

```

    new_indices = np.cumsum(mask)-1
    removed_terms = set()
    for term, old_index in list(six.iteritems(vocabulary)):
        if mask[old_index]:
            vocabulary[term] = new_indices[old_index]
        else:
            del vocabulary[term]
            removed_terms.add(term)
    kept_indices = np.where(mask)[0]
    if len(kept_indices) == 0:
        raise ValueError("After pruning, no terms remain. Try a lower"

```

```

        " min_df or a higher max_df.")
    return X[:, kept_indices], removed_terms

def _count_vocab(self, raw_documents, fixed_vocab):

    if fixed_vocab:
        vocabulary = self.vocabulary_
    else:
        vocabulary = defaultdict()
        vocabulary.default_factory = vocabulary.__len__

    analyze = self.build_analyzer()
    j_indices = []
    indptr = _make_int_array()
    values = _make_int_array()
    indptr.append(0)
    for doc in raw_documents:
        feature_counter = {}
        for feature in analyze(doc):
            try:
                feature_idx = vocabulary[feature]
                if feature_idx not in feature_counter:
                    feature_counter[feature_idx] = 1
                else:
                    feature_counter[feature_idx] += 1
            except KeyError:
                continue

        j_indices.extend(feature_counter.keys())
        values.extend(feature_counter.values())
        indptr.append(len(j_indices))

    if not fixed_vocab:
        vocabulary = dict(vocabulary)
        if not vocabulary:
            raise ValueError("empty vocabulary; perhaps the documents only")

```

```

        " contain stop words")

j_indices = np.asarray(j_indices, dtype=np.intc)
indptr = np.frombuffer(indptr, dtype=np.intc)
values = np.frombuffer(values, dtype=np.intc)

X = sp.csr_matrix((values, j_indices, indptr),
                  shape=(len(indptr)-1, len(vocabulary)),
                  dtype=self.dtype)
X.sort_indices()
return vocabulary, X

def fit_transform(self, raw_documents, y=None):

    if isinstance(raw_documents, six.string_types):
        raise ValueError(
            "Iterable over raw text documents expected, "
            "string object received.")

    self._validate_vocabulary()
    max_df = self.max_df
    min_df = self.min_df
    max_features = self.max_features

    vocabulary, X = self._count_vocab(raw_documents,
                                       self.fixed_vocabulary_)

    if self.binary:
        X.data.fill(1)

    if not self.fixed_vocabulary_:
        X = self._sort_features(X, vocabulary)

    n_doc = X.shape[0]
    max_doc_count = (max_df

```



```

        if isinstance(max_df, numbers.Integral)
        else max_df * n_doc)
min_doc_count = (min_df
    if isinstance(min_df, numbers.Integral)
    else min_df * n_doc)
if max_doc_count < min_doc_count:

    raise ValueError(
        "max_df corresponds to < documents than min_df")
X, self.stop_words_ = self._limit_features(X, vocabulary,
                                           max_doc_count,
                                           min_doc_count,
                                           max_features)

self.vocabulary_ = vocabulary
return X

def transform(self, raw_documents):
    if isinstance(raw_documents, six.string_types):
        raise ValueError(
            "Iterable over raw text documents expected, "
            "string object received.")

    if not hasattr(self, 'vocabulary_'):
        self._validate_vocabulary()

    self._check_vocabulary()

    _, X = self._count_vocab(raw_documents, fixed_vocab=True)
    if self.binary:
        X.data.fill(1)
    return X

def get_feature_names(self):
    self._check_vocabulary()

```

```

        return [t for t, i in sorted(six.iteritems(self.vocabulary_),
                                     key=itemgetter(1))]

def _make_int_array():
    return array.array(str("i"))

class DecisionTreeClassifier(BaseDecisionTree, ClassifierMixin):

    def __init__(self,
                  criterion="gini",
                  splitter="best",
                  max_depth=None,
                  min_samples_split=2,
                  min_samples_leaf=1,
                  min_weight_fraction_leaf=0.,
                  max_features=None,
                  random_state=None,
                  max_leaf_nodes=None,
                  min_impurity_decrease=0.,
                  min_impurity_split=None,
                  class_weight=None,
                  presort=False):

        super(DecisionTreeClassifier, self).__init__(
            criterion=criterion,
            splitter=splitter,
            max_depth=max_depth,
            min_samples_split=min_samples_split,
            min_samples_leaf=min_samples_leaf,
            min_weight_fraction_leaf=min_weight_fraction_leaf,
            max_features=max_features,
            max_leaf_nodes=max_leaf_nodes,
            class_weight=class_weight,
            random_state=random_state,

```

```

        min_impurity_decrease=min_impurity_decrease,
        min_impurity_split=min_impurity_split,
        presort=presort)

def fit(self, X, y, sample_weight=None, check_input=True,
        X_idx_sorted=None):

    super(DecisionTreeClassifier, self).fit(
        X, y,
        sample_weight=sample_weight,
        check_input=check_input,
        X_idx_sorted=X_idx_sorted)
    return self

def predict_proba(self, X, check_input=True):

    check_is_fitted(self, 'tree_')
    X = self._validate_X_predict(X, check_input)
    proba = self.tree_.predict(X)

    if self.n_outputs_ == 1:
        proba = proba[:, :self.n_classes_]
        normalizer = proba.sum(axis=1)[:, np.newaxis]
        normalizer[normalizer == 0.0] = 1.0
        proba /= normalizer
        return proba

    else:
        all_proba = []
        for k in range(self.n_outputs_):
            proba_k = proba[:, k, :self.n_classes_[k]]
            normalizer = proba_k.sum(axis=1)[:, np.newaxis]
            normalizer[normalizer == 0.0] = 1.0
            proba_k /= normalizer
            all_proba.append(proba_k)
        return all_proba

```

```
def predict_log_proba(self, X):  
    proba = self.predict_proba(X)  
    if self.n_outputs_ == 1:  
        return np.log(proba)  
    else:  
        for k in range(self.n_outputs_):  
            proba[k] = np.log(proba[k])  
    return proba
```