**МиНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ**

**УНИВЕРСИТЕТ им. В.Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

**Институт** Энергетики, информационных технологий и управляющих систем

**Кафедра** Программного обеспечения вычислительной техники и автоматизированных систем

**Шифр, название направления, специальности**

09.03.04 Программная инженерия

Разработка программно-информационных систем

(образовательная программа)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
|  | Утверждаю:  Зав. кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «17»\_\_\_\_\_\_\_\_июня\_\_\_\_\_\_2016г. |

Задание

на выпускную квалификационную работу студента (ки)

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*\_\_Адаменко Игоря Игоревича\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1.Вид выпускной квалификационной работы\_\_\_\_\_\_\_\_\_бакалаврская работа\_\_\_\_\_\_\_\_\_\_\_\_

(дипломный проект (работа), бакалаврская работа, магистерская диссертация)

2. Тема работы\_\_\_\_\_\_\_\_Ситуационная советующая система на основе нечёткой логики\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_утверждено приказом по университету от «16»\_\_\_\_\_мая\_\_\_\_\_2016г. №\_\_\_2/577\_\_\_\_\_\_\_\_\_

3. Срок сдачи студентом законченной работы\_\_\_\_15.06.2016\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Исходные данные\_\_механизмы вывода, основанные на прецедентах; алгоритмы вычисления функции принадлежности прецедента и их выбор; ситуационные советующие\_\_\_\_\_\_\_\_  
системы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Содержание выпускной квалификационной работы (перечень подлежащих разработке разделов)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Ситуационные советующие системы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Механизмы вывода, основанные на прецедентах\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Генератор персональных советующих систем\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Алгоритмическая оболочка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Перечень графического материала\_титульный; ситуационная советующая система на основе нечёткой логики; ситуационные советующие системы; генератор персональных советующих систем и его структура; постановка задачи; построение матрицы знаний по прецедентам; алгоритмы; выбор инструментальных средств и подходов; взаимодействие подсистем приложения; структура базы данных\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Консультанты по работе с указанием относящихся к ним разделов

|  |  |  |  |
| --- | --- | --- | --- |
| Раздел | Консультант | Задание выдал  (подпись, дата) | Задание принял  (подпись, дата) |

Дата выдачи задания «18» апреля 2016г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Синюк В. Г.

(подпись руководителя)

Задание принял к исполнению \_\_\_18 апреля 2016г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Адаменко И. И. (подпись выпускника)

**КАЛЕНДАРНЫЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Наименование этапов работы | Срок выполнения  этапов работы | Примечание |
| 1. | Предпроектное обследование | 29.04.16 | выполнено |
| 2. | Проектирование системы | 08.05.16 | выполнено |
| 3. | Программная реализация | 02.06.16 | выполнено |
| 4. | Оформление проекта | 15.06.2016 | выполнено |

Дипломник\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Адаменко И. И.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Синюк В. Г.

**МиНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ**

**УНИВЕРСИТЕТ им. В.Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

**Институт**\_*Энергетики, информационных технологий и управляющих систем*\_\_\_\_\_\_\_\_\_\_\_

**Кафедра***\_Программного обеспечения вычислительной техники и автоматизированных систем\_*

**Шифр, название направления, специальности**

09.03.04 Программная инженерия

Разработка программно-информационных систем

(образовательная программа)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

на тему:

Ситуационная советующая система на основе нечёткой логики

**Дипломник** *Адаменко И. И.*

**Зав. кафедрой** *Поляков В. М.*

**Руководитель** *Синюк В. Г.*

**К защите допустить**

**Зав. кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/**

**«\_\_\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_г.**

**Белгород 20\_\_\_\_\_г**

СОДЕРЖАНИЕ

[Введение 5](#_Toc453648434)

[1. Ситуационные советующие системы 7](#_Toc453648435)

[2. Механизмы вывода, основанные на прецедентах 13](#_Toc453648436)

[2.1. Лингвистическая переменная 13](#_Toc453648437)

[2.2. Матрицы знаний по прецедентам 15](#_Toc453648438)

[2.3. Алгоритмы вычисления функции принадлежности прецедента и их выбор 16](#_Toc453648439)

[2.4. Алгоритм выбора прецедента при наличии ситуационного вектора с количественными координатами 18](#_Toc453648440)

[2.5. Пример 19](#_Toc453648441)

[3. Генератор персональных советующих систем 22](#_Toc453648442)

[3.1. Блок принятия решений 23](#_Toc453648443)

[3.2. Блок оценки состояний 23](#_Toc453648444)

[3.3. Блок выдачи 23](#_Toc453648445)

[3.4. Блок объяснения и адаптации 24](#_Toc453648446)

[3.5. База знаний и данных 24](#_Toc453648447)

[3.6. Блок экспертного опроса 24](#_Toc453648448)

[3.7. Режимы ГПСС 24](#_Toc453648449)

[4. Алгоритмическая оболочка 27](#_Toc453648450)

[4.1. Платформа 27](#_Toc453648451)

[4.2. Язык программирования 27](#_Toc453648452)

[4.3. Среда разработки 29](#_Toc453648453)

[4.4. Система контроля версий 30](#_Toc453648454)

[4.5. Формат хранения данных 32](#_Toc453648455)

[4.5.1. Объект 33](#_Toc453648456)

[4.5.2. Массив 34](#_Toc453648457)

[4.5.3. Значение 34](#_Toc453648458)

[4.5.4. Обоснование выбора 35](#_Toc453648459)

[4.5.5. Структура данных 35](#_Toc453648460)

[4.6. Проектирование и оценка подходов к реализации 36](#_Toc453648461)

[4.6.1. MVC 36](#_Toc453648462)

[4.6.2. BEM 37](#_Toc453648463)

[4.7. Структура программы 39](#_Toc453648464)

[4.7.1. Основные структурные единицы 39](#_Toc453648465)

[4.7.2. Зависимости структурных единиц 41](#_Toc453648466)

[4.8. Работа программы 41](#_Toc453648467)

[4.9. Тестирование программы 45](#_Toc453648468)

[Заключение 52](#_Toc453648469)

[Список литературы 53](#_Toc453648470)

[Приложение А 55](#_Toc453648471)

[Приложение Б 60](#_Toc453648472)

# Введение

В современном мире существует огромное количество сложных объектов и ситуаций, управление которыми невозможно с использованием только количественной информации. Необходимо также задействовать качественную, т. н. смысловую информацию.

Причины для изучения логико-лингвистических моделей возникли в момент попытки автоматизации объектов столь сложной природы, для которых существующие методы оказались непригодными или малоэффективными. И тому есть несколько причин [1]:

1. Между параметрами, от которых зависит процесс управления, не получается установить точных количественных зависимостей.
2. Далеко не все цели управления объектом (ситуацией) могут быть выражены количественно.
3. Те способы для описания объектов и их процессов, которые существуют, приводят к конструкциям, с которыми сложно, а то и вовсе невозможно работать из-за их громоздкости.
4. Процесс управления — это многошаговый процесс, причём следующий шаг не всегда известен и однозначно определён заранее.

К тому же, если расширять понятие «объект управления», включая в него не только объекты, участвующие в технологических процессах, но и, например, в социальных, то появляются ещё причины, такие как невозможность строгого формулирования цели существования объекта, его эволюция во времени и сложный состав.

Тот факт, что процесс управления — многошаговый (п. 4) накладывает особенные ограничения, связанные с объёмом вычислений. Поскольку следующий шаг заранее не известен, очевидным решением является перебор всех возможных следующих шагов, а затем следующих для них и т. д. Таким образом практически невозможно проанализировать влияние каждого из этих шагов на принимаемое в итоге решение. Для этого и создаются ситуационные советующие системы, использующие механизмы ввода, основанные на прецедентах.

# 1. Ситуационные советующие системы

В связи с технологическим развитием современного мира, системы нацеленные на поддержку принятия решений становятся всё более актуальными. Если не так давно они использовались только при решении каких-либо технологических задач, частично связанных со сложными количественными вычислениями, а частично — с анализом качественных характеристик текущей ситуации, то сейчас и обычному человеку доступны вычислительные мощности, которые могут использоваться (и активно используются) в повседневной жизни. Это может быть как рекомендательная система в онлайн-магазине, основанная на качественных отзывах покупателей («Мне понравилось, что..», «Мне не понравилось, что..» и т. д.), так и система поддержки водителя автомобиля, учитывающая всевозможные количественные (скорость, количество топлива, температура двигателя) и качественные («мастерство» водителя) характеристики.

Название этих систем вполне соответствует действительности: они по запросу пользователя выдают «советы» о рекомендуемых последующих действиях, которые приведут к обозначенной заранее цели, причём, советы, основанные не на простых вычислениях в идеальном мире, а на опыте реальных людей, авторитетов в своей области.

Как, опять же, следует из названия, советующая система не заменяет человека, она только подсказывает ему, какое действие он должен предпринять. Это связано с тем, что право выбора должно оставаться за человеком, который так или иначе часто более опытен, чем система, однако его скорость реакции во много раз медленнее. Поэтому такие системы не включаются в контур управления, хотя бывают и исключения. Например, автопилот в самолёте.

И то, в таком случае, самой советующей системы в контуре управления нет, есть только механизм, который получает данные от неё и на их основе прини-

мает решение. А то, как включается этот механизм — вручную человеком или автоматически в экстренной ситуации, это уже зависит от потребностей конкретной системы.

Таким образом, в 1990 году А. Н. Мелихов, Л. С. Берштейн и С. Я. Коровин подготавливают монографию [1], с целью развития ситуационного подхода к принятию решений на основе нечёткой логики, выполняя для этого анализ различных нечётких моделей, которые могут быть использованы в системах распознавания образов, диагностических, экспертных и советующих системах, применяемых в условиях неполноты, недоопределённости и нечёткости исходной информации. Они рассматривают вопросы программной и аппаратной реализации предложенных алгоритмов, подкрепляя тем самым содержание монографии решением практических задач для различных областей техники. Рецензентом работы является доктор технических наук Д. А. Поспелов, благодаря которому появилось понятие «логико-лингвистическая модель».

Возникает естественный вопрос, откуда система будет брать данные для определения того, что лучше сделать в текущей ситуации? Если мы рассматриваем какой-то объект, для управления которым уже есть опытный оператор, то модель управления этим объектом уже создана. Она существует в виде правил в памяти оператора, которые необходимо представить в приемлемой для ЭВМ форме. Однако это не столь тривиальная задача. Человеку удобнее оперировать качественными характеристиками, нежели количественными («повернуть чуть-чуть», «наклонить сильнее», «очень близко», «медленно» и пр.), что доказано на практике [3, 4]. Для формализации этих нечётких понятий используется нечёткая логика.

Основная часть советующей системы с нечёткой логикой (ССНЛ) является нечёткая модель управления. Структура этой модели состоит из трёх крупных блоков [2], изображённых на рисунке 1.1. Основным является блок принятия решений (БПР), который получает информацию из блока оценки состояний (БОС), обрабатывает её и передаёт результат в блок выдачи управляющих взаимодействий (БВУВ).

01

Рисунок 1.1 — Архитектура ССНЛ

БОС получает информацию от оператора и строит на её основе формализованное описание текущей ситуации. Чтобы оценить семантику нечётких понятий используются понятие лингвистической переменной, введённой Л. Заде в [5]. Это своего рода такая форма организации экспертной информации, на основе которой и производится семантическая идентификация нечётких категорий. Когда блок получает на вход какую-то качественную информацию (нечёткое понятие), он ставит ему в соответствие нечёткое множество на определённой предметной шкале, которое или строится на основе экспертного опроса оператора, или выводится из уже имеющихся нечётких множеств [5, 6]. Стоит заметить, что чёткая, количественная информация также получает в соответствие нечёткое множество. Таким образом система оперирует полностью нечёткими понятиями.

По полученным данным БОС строит формализованное описание текущей ситуации, своего рода композицию нечётких множеств, и передаёт его в БПР. БПР же на основе нечётких логических заключений определяет, какое управляющее решение наиболее подходит в данной ситуации и передаёт его в БВУВ, который выполняет действия, обратные БОС, т. е. переводит имеющиеся нечёткие понятия в форму, в которой они будут понятны пользователю, при необходимости аппроксимируя и интерпретируя их на уровне лингвистики [7, 8]. Подробнее о блоках будет описано в следующих разделах данной работы.

Главными проблемами, решаемыми при построении системы принятия решений с нечёткой логикой, являются: сопоставление описаний состояний объекта принятия решения с условиями истинности продукции и определение последовательности просмотра и анализа продукции при выводе решений. В зависимости от методов решения этих проблем советующие системы разделяются на два класса: «ситуация — действие» (С-Д) и «ситуация — стратегия управления — действие» (С-СУ-Д).

Решение первой проблемы одинаково для обоих классов систем: используя эталонные состояния, представленные в виде нечётких ситуаций, а также текущее состояние (тоже как нечёткую ситуацию) с помощью специальных мер близости определяется необходимая эталонная ситуация (методом ближайшего соседа).

А решение второй проблемы различно, так как продукции описаны в системах по-разному. В С-Д-системах продукции описаны явно, в виде нечёткой базы знаний, а условия их истинности задаются эталонными состояниями (т. е. эталонными нечёткими ситуациями), причём каждая продукция также содержит ещё и управляющее решение, которое выдаётся после процесса вывода решения. В процессе вывода решения сопоставляется описание текущей ситуации со всеми эталонными, определение продукции с наиболее подходящей эталонной ситуации и нахождение соответствующего управляющего решения. При этом последовательность просмотра продукций, как правило, неизменна.

Если формализовывать описанное выше, то продукционная система в С-Д-системах ставит в соответствие каждой ситуации из определённого набора ситуаций , некоторое управляющее решение . Набор содержит эталонные ситуации, и в отличие от набора типовых ситуаций , не содержит нечётко равных при определённом пороге равенства ситуаций. Это уменьшает размерность продукционной системы и не снижает эффективности модели управления в пределах достоверности, ограничиваемых порогом равенства.

В системах С-СУ-Д же продукции не задаются в явном виде, а выводятся с помощью нечёткой ситуационной сети (НСС), которая является ни чем иным, как нечётким взвешенным графом переходов по эталонными ситуациям. С помощью нечёткого маршрута между текущей и необходимой ситуацией определяется набор продукций и последовательность их просмотра. Необходимая (т. н. целевая) ситуация выбирается с помощью специальной продукционной системы, или же с помощью НСС. За счёт использования НСС полученная система устойчива к резким изменениям условий управления, т. е. система получается очень гибкой.

Так как целевая ситуация в модели С-СУ-Д может задаваться явно, то в ней используется продукционная система типа «ситуация — ситуация». Также, поскольку целевая ситуация может определяться исходя из анализа степеней предпочтения управляющих решений, для её постановки может быть построена НСС, вершины которой соответствуют эталонным нечётким ситуациям, дуги взвешены управляющими решениями, необходимыми для перехода по ситуациям, и степенями предпочтения этих решений.

Степени предпочтения управляющих решений или неизменны в каждой ситуации (т. к. определяются экспертным опросом), или же некоторым образом зависят от ситуации, и тогда для их определения используется продукционная система типа «ситуация — предпочтение решений» (С-ПР). Управляющее решение, соответствующее текущей ситуации, представляет собой последовательность решений, необходимых для перехода от текущей ситуации к целевой по оптимальному в некотором смысле маршруту в НСС, который и называется стратегией управления. Таким образом, вывод решения в модели С-СУ-Д разбивается на два этапа: постановка цели (целевой ситуации) и построение стратегии управления. Ситуация с возможными в ней управляющими решениями по сути дела представляет собой продукцию системы С-Д. Можно сказать, что стратегия управления задаёт последовательность «просмотра» продукций в продукционной системе «ситуация — действие», соответствующую оптимальному переводу объекта управления в целевое состояние.

Вывод

В данном разделе была представлена основная теоретическая информация, связанная с ситуационными советующими системами и их классами. Также в этом разделе была обоснована актуальность исследования этой темы и потребность генераторов ситуационных советующих систем в повседневной жизни.

# 2. Механизмы вывода, основанные на прецедентах

В повседневной жизни и в работе часто возникают такие ситуации, сложность которых не позволяет принять решение мгновенно, однако, по подобным ситуациям уже может быть база данных с принятыми решениями, которые можно назвать успешными. В качестве примера можно привести работу менеджера в IT-компании, который управляет штатом сотрудников разной направленности и подготовки. Во время возникновения очередной проблемной ситуации, он может обратиться к своему опыту (т. е. к БД с успешными решениями) и понять, кому нужно назначить задачу, чтобы она была решена как можно быстрее. Но ситуация вполне может быть настолько сложной и запутанной, что менеджер не сможет найти решение со стопроцентной уверенностью. В таком случае ему поможет система, которая по имеющимся у него данным о прошлых ситуациях сможет разрешить текущую.

Итак, механизмы вывода применяются в сложных ситуациях, по которым есть прецеденты их успешного разрешения. Однако, у этого подхода есть свои трудности. Одна из них — это правильный подбор координат ситуационного выбора, описывающего текущую ситуацию, как по их количеству, так и по форме представления каждой из них. Полноту описания ситуационного вектора и связь конкретного вектора с конкретным прецедентом определяется с помощью экспертов — носителей знания об этом. Координаты ситуационного вектора же — это лингвистические переменные.

## 2.1. Лингвистическая переменная

Л. Заде определил [5] лингвистическую переменную как переменную, принимающую свои значения из заданного множества высказываний естественного языка, каждое из которых называется термом.

В качестве примера лингвистической переменной можно привести следующую: *«температура» = {«очень низкая», «низкая», «средняя», «высокая», «очень высокая»}*. В это примере *«температура»* — это лингвистическая переменная, а в перечисленные в фигурных скобках значения — термы этой лингвистической переменной.

При работе с лингвистическими переменными, каждый терм из множества их значений соотносится с нечетким множеством, которое представляется в виде универсального множества и функции принадлежности каждого элемента этому множеству. Значениями этой функции принадлежности являются числа, принадлежащие [0; 1], которые задают степень принадлежности элемента универсальному множеству, т. е. являются количественным выражением терма лингвистической переменной.

В качестве примера рассмотрим описанную выше лингвистическую переменную *«температура»*. Возьмём терм *«очень низкая»*. Универсальное множество в этом случае будет равно [–273,15; 300]. Функция принадлежности нечеткого множества представлена на рисунке 2.1.

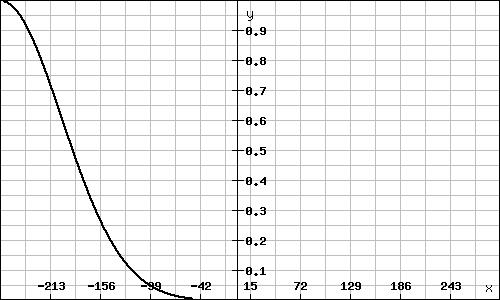


Рисунок 2.1 — Функция принадлежности нечёткого множества терма «очень низкая»

Стоит отдельно отметить, что универсальное множество, как и функции принадлежности, определяются с помощью исследования экспертами соответствующей предметной области, связанной с этой задачей.

## 2.2. Матрицы знаний по прецедентам

Пусть мы имеем ситуацию, которая описывается вектором с координатами , где каждая координата — это лингвистическая переменная со своим набором термов , а также набор успешных разрешений ситуаций, в которых каждая лингвистическая переменная принимает одно из своих возможных значений. Допустим, что помимо этого накоплено множество прецедентов — — каждый из которых связан со своим набором ситуационных векторов, при которых он являлся решением ситуации. Имея всё описанное выше, составим матрицу соответствия прецедентов наборам ситуационных векторов (таблица 2.1).

Таблица 2.1 — Координаты ситуационных векторов и прецеденты

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Координаты ситуационного вектора | | | | | Прецедент |
|  | .. |  | .. |  |
|  |  | .. |  | .. |  |  |
| .. | .. | .. | .. | .. | .. |
|  |  | .. |  | .. |  |
| .. | .. | .. | .. | .. | .. | .. |
|  |  | .. |  | .. |  |  |
| .. | .. | .. | .. | .. | .. |
|  |  | .. |  | .. |  |
| .. | .. | .. | .. | .. | .. | .. |
|  |  | .. |  | .. |  |  |
| .. | .. | .. | .. | .. | .. |
|  |  | .. |  | .. |  |

В этой таблице каждая строка соответствует конкретному ситуационному вектору и прецеденту, который успешно реализовался в прошлом.

Строки каждого блока прецедента пронумеруем таким образом, чтобы первое число соответствовало номеру прецедента ( и т. д.), а второе — порядковому номеру ситуационного вектора.

Таким образом мы получили систему логических высказываний, каждое из которых соответствует конструкции *«если …, то …, иначе …»*. К примеру, для строки мы получим такое высказывание:

|  |  |
| --- | --- |
| *«Если и … и и … и , то , иначе <аналогичное высказывание для следующей строки>»* | (1) |

Полученная матрица и является матрицей знаний по прецедентам.

## 2.3. Алгоритмы вычисления функции принадлежности прецедента и их выбор

Как описано в [10], предположим, что алгоритм определения функции принадлежности прецедента , есть нечёткое множество на универсальном множестве , где — универсальное множество, на котором заданы термы лингвистической переменной , а — декартово произведение таких универсальных множеств.

Каждая строка матрицы знаний по прецедентам (т. е. каждое высказывание типа (1)) есть нечёткое отношение соответствующих нечётких множеств. Например, для (1) это будет:

.

Основываясь на [6, 10] заключаем, что функция принадлежности нечёткого множества для этого нечёткого отношения будет выглядеть следующим образом:

*,*

где символ обозначает операцию выбора минимального (min).

Таким образом, если проанализировать все логические высказывания из таблицы 2.1, относящиеся к прецеденту , можно заметить, что они есть ни что иное, как объединение соответствующих нечётких множеств. А функция принадлежности этого объединения (т. е. функция принадлежности прецедента ) согласно [10] будет выражаться следующим образом (знак обозначает взятие максимального (max)):

Для наглядности отобразим описанные выше формулы на примере одного прецедента, данные о котором соберём в таблицу 2.2 представленную ниже.

Таблица 2.2 — Координаты ситуационных векторов, min, max и прецеденты

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | Координаты ситуационного вектора | | | | | min | max | d |
|  | .. |  | .. |  |
| .. | .. | .. | .. | .. | .. | .. | .. | .. |
|  |  | .. |  | .. |  |  |  |  |
| .. | .. | .. | .. | .. | .. | .. |
|  |  | .. |  | .. |  |  |
| .. | .. | .. | .. | .. | .. | .. |
|  |  | .. |  | .. |  |  |
| .. | .. | .. | .. | .. | .. | .. | .. | .. |

Если описывать представленный выше алгоритм определения функции принадлежности прецедента более формально, то можно разбить его на три последовательных шага.

**Шаг первый.** Фиксируем какую-то произвольную точку .

**Шаг второй.** Определяем значение функции принадлежности в этой точке для каждого блока матрицы знаний, соответствующего , как описано в таблице 2.2.

Стоит отметить, что для выбранной точки блок матрицы в таблице 2.2 является уже числовым, т. к. каждый терм мы заменяем на значение его функции принадлежности —  — подсчитанное в .

**Шаг третий.** Выполняем последовательно операции нахождения минимального (над числами в строках) и максимального (над найденными ранее минимальными). Полученное в результате поиска максимального число является значением функции принадлежности в выбранной в первом шаге точке .

## 2.4. Алгоритм выбора прецедента при наличии ситуационного вектора с количественными координатами

Во многих повседневных ситуациях помимо качественных характеристик встречаются ещё и количественные. Чаще всего это бывают показания каких-либо приборов (температура, влажность и пр.), но это могут быть и оценки самого эксперта. В таком случае функция принадлежности определяется конкретно в этой точке ситуационного вектора.

Дело в том, что, когда мы имеем дело с ситуационным вектором с количественными координатами [12] (т. е. все координаты измерены по числовым шкалам), для выбора оптимального прецедента не нужно определять функции принадлежности для всех точек универсума. Достаточно только рассчитать её для текущих точек, а для этого придётся воспользоваться однократно алгоритмом вычисления функции принадлежности прецедента, описанным в предыдущем подпункте этой главы. В качестве кортежа координаты текущего ситуационного вектора.

Поскольку после вычислений мы получим число , которое есть ни что иное, как степень принадлежности точке , то исходя из описанного в таблице 2.2. наиболее предпочтительным будет тот прецедент , для которого: .

В итоге, в отличие от [10], если механизм вывода реализован таким образом, что данные могут вводиться не только с клавиатуры пользователя, то их можно вводить в алгоритм предпочтения прецедентов напрямую из устройства, выдающего такое значение, минуя пользователя.

## 2.5. Пример

Предположим, что мы имеем дело с ситуацией, которая описывается вектором с координатами , которые представлены следующими лингвистическими переменными:

* — средняя скорость автомобиля;
* — дальность видимости дороги водителем;
* — средний тормозной путь.

Допустим также, что для класса подобных ситуацией наблюдалось два успешных разных прецедента: .

Лингвистические термы принимают одно из трёх значений: , и каждый из них представлен на базовом универсальном множестве пятибалльной шкалы, а функции принадлежности выглядят так, как на рисунке 2.2. А матрица знаний для этой ситуации представлена в таблице 2.3.

Таблица 2.3 — Матрица знаний

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Координаты ситуационного вектора | | | Прецедент |
|  |  |  |
|  |  |  |  |  |
|  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |

Рисунок 2.2 — Функции принадлежности

Пусть координаты ситуационного вектора для наблюдаемой ситуации равны: . Тогда по алгоритму выбора предпочтительного прецедента получим данные, описанные в таблице 2.4.

Таблица 2.4 — Данные для конкретного ситуационного вектора

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Координаты ситуационного вектора | | | min | max | d |
|  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |

Исследуя представленную выше таблицу, можем прийти к следующему выводу: наиболее предпочтительным прецедентом для разрешения имеющейся ситуации с ситуационным вектором является прецедент , т. к. в предпоследнем столбце, мы имеем 0 для и 0,5 . Выбирая большее, получаем ответ.

Вывод

В данном разделе были описаны основные понятия и алгоритмы, необходимые для работы с ситуационными советующими системами и для их непосредственной разработки. Это абстрактное необходимое и достаточное знание, которое далее можно применять в конкретной предметной области. Применение так же было проиллюстрировано примером с конкретными данными.

# 3. Генератор персональных советующих систем

В этой главе будет рассмотрена основная структура генератора персональных советующих систем (ГПСС). Разработка генераторов таких систем вызвана необходимостью облегчения создания советующих систем для разных областей применения. Суть их сводится к тому, что создаётся «пустая» советующая система, которая затем с помощью эксперта и имеющихся данных настраивается под конкретную сложную ситуацию, в которой в данных момент сложно выбрать необходимый прецедент.

Заполнение данными происходит в режиме экспертного опроса, причём, предполагается, что пользователь, который участвует в экспертном опросе, не знаком с тем, как система устроена внутри, и потому интерфейс общения с ним построен исключительно на естественном языке, т. к. этот пользователь должен комфортно передавать свои данные, не отвлекаясь на излишние вычисления.

ГПСС состоит из шести основных блоков:

* Блок оценки состояний (БОС);
* Блок принятия решений (БПР);
* Блок выдачи (БВ);
* Блок объяснения и адаптации (БОА);
* База знаний и данных (БД);
* Блок экспертного опроса (БЭО).

Все они настраиваются экспертом (оператором) с помощью БЭО. Взаимодействие систем отображено на рисунке 3.1 (пунктирные линии — взаимодействие в режиме настройки, сплошные — в режиме эксплуатации).

Разберём работу каждого блока отдельно.

03

Рисунок 3.1 — Структура ГПСС

## 3.1. Блок принятия решений

Главной компонентой ядра системы является этот блок. В основу его функционирования положен принцип определения по системе продукций необходимых при данной входной ситуации управляющих решений. Продукционная система ставит в соответствие каждой ситуации из определённого набора ситуаций, характеризующего все возможные состояния объекта управления, некоторое управляющее решение. Ситуации, входящие в набор, называются эталонными. В отличие от набора типовых ситуаций, набор эталонных ситуаций не содержит нечётко равных при заданном пороге равенства ситуаций. Это способствует уменьшению размерности продукционной системы и не снижает эффективности модели управления в пределах достоверности, ограничиваемых порогом равенства [2].

## 3.2. Блок оценки состояний

Этот блок строит формализованное описание ситуации, возникшей на объекте управления, с помощью поступающей на вход информации.

## 3.3. Блок выдачи

В данном блоке осуществляется переход от внутренней формы задания управляющих решений к внешней [14].

## 3.4. Блок объяснения и адаптации

Блок объяснения и адаптации по запросу пользователя системы производит объяснение «мотивов» выработки того или иного решения. На любом этапе объяснения пользователь имеет возможность по своему усмотрению корректировать содержимое БД, подстраивая таким образом советующую систему в соответствии со своими представлениями о правильных решениях.

## 3.5. База знаний и данных

БД состоит из нескольких основных частей:

* характеристика признаков и лингвистических переменных;
* характеристика управляющих решений;
* зависимости между решениями;
* ограничения на применение управляющих решений;
* соответствие между конфликтными и целевыми эталонными ситуациями;
* описание некоторых параметров советующей системы.

## 3.6. Блок экспертного опроса

Данные блок предназначен для создания и настройки блока оценки состояний, блока принятия решений и операционной части самого блока экспертного опроса, а также заполнения БД в интерактивном режиме.

## 3.7. Режимы ГПСС

Во время использования ГПСС, как уже говорилось ранее, она обычно работает в двух режимах: сперва работа происходит в режиме экспертного опроса, затем же в режиме эксплуатации. Это несёт в себе небольшой «конструктивный» недостаток, поскольку при больших объёмах данных первый режим (экспертный опрос) становится долгим и утомительным для пользователя, чем негативно влияет на работу системы в целом [2]. Современные методы построения ПО позволяют объединить два режима в один и, при помощи хорошо разработанного редактора знаний, вносить изменения в экспертную информацию уже в процессе работы системы.

Если рассматривать работу ГПСС в одном режиме, описанном выше, то пунктирные линии на рисунке 3.1, ранее обозначавшие взаимодействия компонентов при экспертном опросе, можно воспринимать как постоянное взаимодействие блоков, вызванное корректировками пользователя. В таком случае, мы всё равно имеем два режима работы, которые теперь постоянно сменяют друг друга, по желанию пользователя, но незаметно для него.

Когда пользователь вносит изменения в экспертную информацию, он конфигурирует особенности объекта управления и ограничения на принимаемые решения, необходимые для создания генератором нечёткой ситуационной модели управления объектом. В этом случае активным является только блок экспертного опроса, получающий и перерабатывающий информацию для всех остальных блоков системы. Он получает данные о пороге сравнения, признаках, лингвистических переменных, управляющих решениях и зависимостях между ними. Затем переформировывает базовую шкалу значений признаков и перестраивает функции принадлежностей.

В режиме эксплуатации, то есть, когда пользователь не изменяет экспертную информацию, активным в ГПСС является ядро системы (рисунок 3.1), которое, выполняет функции управления системой. Работа в этом режиме также происходит в диалоге с пользователем. Ему задаются вопросы с целью получения описания состояния объекта управления. Поиск стратегии управления в текущей ситуации возлагается на блок принятия решений. Также пользователь может перевести генератор в режим объяснения и адаптации, при котором будет задействован БОА.

Подобная схема ГПСС универсальна и может быть применена для создания советующих систем любой сложности, а при правильном конструировании ПО также позволяет создавать системы, быстро адаптирующиеся к изменяющимся условиям функционирования.

Вывод

В этом разделе мы обосновали необходимость построения генератора персональных советующих систем, а также описали его основную структуру, режимы и конструктивные недостатки при стандартном построении, предложив более оптимальное решение. Используя эту структуру можно перейти к построению алгоритмической оболочки на её основе.

# 4. Алгоритмическая оболочка

## 4.1. Платформа

Разрабатываемое ПО будет веб-приложением, которое работает при помощи браузера. Тем самым достигается переносимость и независимость приложения от используемой платформы, единственным требованием к которой является возможность запуска любого современного браузера (Chrome, Firefox, Opera, Internet Explorer, Edge, Я.Браузер, Safari и т. д.) или использование встроенного в операционную систему (если речь о мобильных устройствах).

Другим преимуществом такого выбора является возможность удалённого запуска программы на мобильных устройствах и прочих медиа-устройствах, имеющих доступ в сеть Интернет.

Также при разработке веб-приложения его можно запускать как локально, таки «выкладывать» для общего пользования в сеть Интернет. Таким образом достигается наиболее лёгкий доступ будущего пользователя к ПО, поскольку ему не нужно иметь дело с файлами программного продукта напрямую.

## 4.2. Язык программирования

Для разработки программного продукта выбран язык программирования JavaScript, а также языки гипертекстовой и стилевой разметки HTML и CSS.

JavaScript — прототипно-ориентированный сценарный язык программирования [9], разработанный Бренданом Айком в компании Netscape в 1995 году.

JavaScript изначально разрабатывался как язык для исполнения программных сценариев на веб-страницах в браузере Netscape Navigator.

Впоследствии язык развился до масштабов промышленной разработки и на данный момент на нём успешно программируются как клиентские части веб-приложений, так и серверные, а также прикладные программы, не связанные с веб-технологиями, и микроконтроллеры.

Важной особенностью языка JavaScript является то, что при наличии браузера на компьютере, пользователь уже имеет встроенный в него интерпретатор, с помощью которого может запускать код на этом языке. Таким образом, разработанные на нём программы можно запускать практически на любом современном медиа-устройстве. Помимо этого, т. к. интерпретатор содержится внутри браузера, он не может навредить системе пользователя без его ведома, поскольку браузер не предоставляет ему программный интерфейс для совершения опасных действий. Потому ещё одним достоинством работы JavaScript является безопасность использования программ, написанных на нём, для пользователя.

Одним из недостатков языка JavaScript является его скорость. К сожалению, в силу конструктивных особенностей, математические операции в нём исполняются медленнее, чем во многих прикладных языках программирования — C, C++, Java и пр. Однако, для данной задачи это не играет большой роли, поскольку используемые алгоритмы являются вычислимыми и не являются ресурсозатратными. К тому же исполнение программ в современных браузерах сильно оптимизировано на низком уровне, а потому не является проблемой даже для маломощных ЭВМ.

Помимо того, что язык может запускаться в браузере, из-за его бурного развития появились инструменты, позволяющие скомпилировать имеющееся веб-приложение в нативное приложение для конкретной операционной системы. То есть, на выходе будет .exe, .app или бинарный файл, который можно запустить, соответственно, на Windows, Mac OS X или Linux-совместимой платформе. Это удобно для тех случаев, когда установка браузера на машину является явно излишним действием.

Также на конференции I/O 2016 компания Google представила новый подход к работе веб-приложений — Progressive Web Apps. При использовании этого подхода можно добиться того, что на мобильных платформах веб-приложение будет запускаться как полностью соответствующее всем гайдлайнам этих платформ. Т. е., например, на Android при запуске приложения не будет появляться адресная строка, будет отображаться экран загрузки и пр., и потому пользователь вообще не сможет отличить веб-приложение от нативного Android-приложения.

Таким образом, выбранный язык является идеальным для написания кроссплатформенных приложений.

## 4.3. Среда разработки

Поскольку JavaScript — не принадлежащий какой-либо корпорации язык, разрабатывать на нём можно в любой среде, начиная от Блокнота (notepad.exe в Windows) и заканчивая полноценными IDE (WebStorm, Intellij IDEA).

Это ПО разрабатывалось в Sublime Text 3 — кроссплатформенном текстовом редакторе [15]. Первая версия этого редактора появилась в 2008-ом году и изначально не была ориентирована под редактирование исходных кодов программ, а также работала только под управлением операционной системы Windows. Со временем, из-за возможностей редактора он начал использоваться всё чаще как среда разработки, а потому его создатель (Джон Скиннер) добавил в него множество полезных функций, облегчающих работу с исходным кодом:

* миникарта — всё содержимое текущего файла отображается в правой части экрана и при клике на неё можно перейти к необходимому участку;
* множественная правка — возможность расстановки нескольких указателей в тексте, параллельное выделение, ввод и удаление символов, что облегчает редактирование значений и названий переменных, функций и прочих участков исходного кода;
* автодополнение — в зависимости от используемого языка редактор при вводе символов пользователем предлагает ему возможные «окончания» ввода, или же автоматически подставляет наиболее подходящий при нажатии на клавишу Tab;
* подсветка синтаксиса — для лучшей читаемости кода и ориентирования программиста в нём, разные его части отрисовываются в окне редактора разными цветами;
* менеджер пакетов — позволяет расширять возможности редактора дополнительными подпрограммами, выполняющими какие-либо узконаправленные действия (работа с кодом, файлами и пр.).

## 4.4. Система контроля версий

При разработке ПО одним из важных пунктов является выбор системы контроля версий. Она необходима для того, чтобы можно было безопасно разрабатывать новые функции в программе и исправлять старые, при необходимости перемещаясь по истории разработки вперёд и назад во времени.

Стоит понимать, что при разработке ПО нельзя просто создавать папки/проекты с разными версиями на жёстком диске и при необходимости переключаться с одного на другой. Эта модель работает при разработке небольшой программы, на которую уходит мало времени. Если же исходный код ПО хранится более, чем в одном файле, или разработка происходит в несколько этапов, то в этом случае помогают системы контроля версий. Они позволяют:

* сравнивать версии ПО написанные в разное время и разными людьми;
* разрабатывать параллельно несколько версий ПО, при необходимости объединяя или разъединяя их между собой, для разработки новых возможностей;
* получать в любой момент всю историю изменений, с описанием действий и затронутых файлов;
* в любой момент времени отменить любые изменения, внесённые в исходный код программы и т. д.

При разработке этого ПО использовалась система контроля версия Git, разработанная Линусом Торвальдсом в рамках проекта Linux в 2005-ом году. Основные достоинства этой системы контроля версий по сравнению с остальными:

* скорость работы;
* простота дизайна;
* поддержка нелинейной разработки (с помощью инструмента, называемого «ветками»);
* полная распределённость;
* эффективная работа даже с большими проектами (т. к. изначально система создавалась для работы над ОС Linux).

Главное отличие Git от других систем контроля версий в том, что он не хранит каждую версию изменённого файла в репозитории, а сохраняет «слепки» файловой системы, и если какой-то файл не изменился, то он хранит только ссылку на его версию в предыдущих слепках. Это позволяет, во-первых, работать с файлами очень быстро, а во-вторых, сократить размер репозитория до минимума. Так, разработанное в рамках этого дипломного проекта ПО занимает на жёстком диске около 5 Мб, а репозиторий с изменениями всего 90 Кб. Если бы использовалась другая система контроля версий, то размер репозитория измерялся бы десятками мегабайт.

При разработке использовался следующий подход:

* основной код программного продукта лежит в ветке master;
* если необходимо разработать новый функционал, то создаётся ветка с префиксом feature/ и названием этого функционала на английском языке;
* аналогично, если необходимо исправить какую-то неявную ошибку, исправление которой займёт много времени, создаётся ветка с префиксом bugfix/ и названием ошибки на английском языке.

Вся разработка велась с помощью проектного менеджера GitHub и его программного обеспечения для Windows. Это позволяло при сравнении файлов пользоваться визуально понятными интерфейсами, а также с помощью GitHub Issues планировалась разработка нового функционала. Интерфейс проектного менеджера представлен на рисунке 4.1.

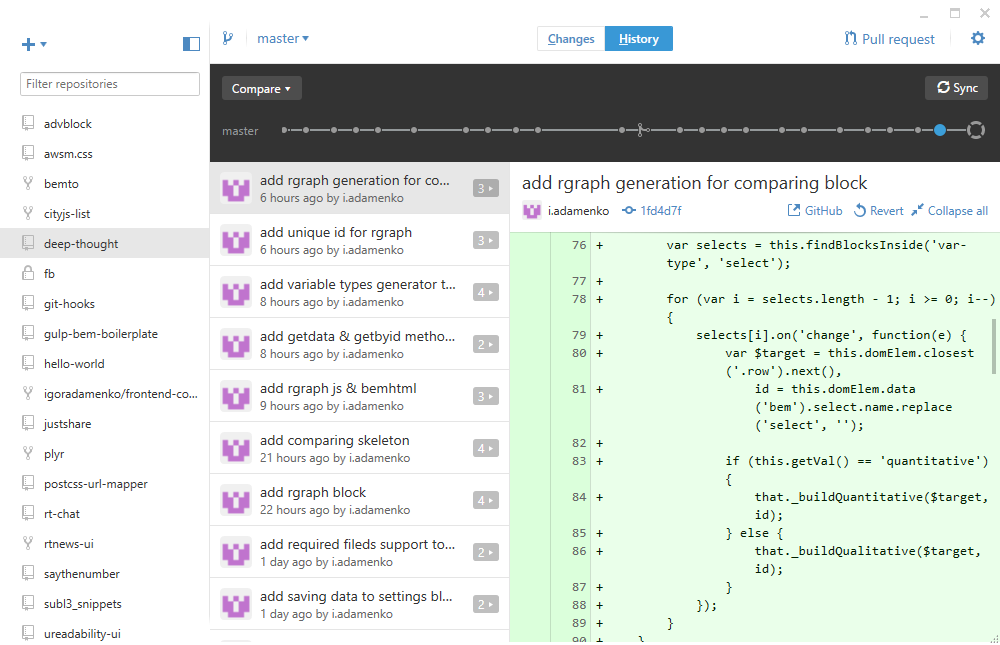


Рисунок 4.1 — Клиент GitHub для Windows

## 4.5. Формат хранения данных

Ключевыми данными на проекте являются данные о лингвистических переменных их термах и значениях нечётких множеств. Поскольку разрабатывается веб-приложение, то в качестве формата хранения данных можно выбрать практически любой. Разница лишь в том, что некоторые форматы можно хранить прямо на клиенте, а некоторые только на сервере.

Для облегчения взаимодействия пользователя с программой мы будем хранить данные на клиентской стороне, т. е. в браузере. Это позволит сделать программный продукт независимым от качества текущего интернет-соединения пользователя и, к тому же, позволит хранить данные в удобном для JavaScript виде.

JSON (JavaScript Object Notation) — это простой формат обмена данными, удобный для чтения и написания как человеком, так и компьютером. JSON — это текстовый формат, полностью независимый от языка реализации, но он использует соглашения, знакомые программистам C-подобных языков, таких как C, C++, C#, Java, JavaScript, Perl, Python и многих других. Эти свойства делают JSON идеальным языком обмена данными, причём, как между сервером и клиентом, так и между подпрограммами в самой программе.

Основными структурами данных в JSON являются:

* коллекции пар ключ/значение — структура данных, которая в других языках программирования может называться объектом, записью, структурой, словарём, хэшом, именованным списком или ассоциативным массивом;
* упорядоченный список значений — структура данных, которая также носит такие названия, как: массив, вектор, список и последовательность.

Поскольку это универсальные структуры данных и почти все современные языки программирования поддерживают работу с ними, логично предположить, что это делает JSON независимым от выбранного языка программирования.

Рассмотрим подробнее нотацию JSON.

### 4.5.1. Объект

Объект — это неупорядоченный набор пар ключ/значение. Его описание начинается с открывающей фигурной скобки и заканчивается закрывающей фигурной скобкой. Каждое имя (ключ) сопровождается двоеточием, а сами пары ключ/значение разделяются запятой. Синтаксическая диаграмма этой структуры данных представлена на рисунке 4.2.



Рисунок 4.2 — Синтаксическая диаграмма объекта

### 4.5.2. Массив

Массив — это упорядоченный набор значений. Его описание начинается с открывающей квадратной скобки и заканчивается закрывающей квадратной скобкой. Значения разделены запятой. Синтаксическая диаграмма этой структуры данных представлена на рисунке 4.3.

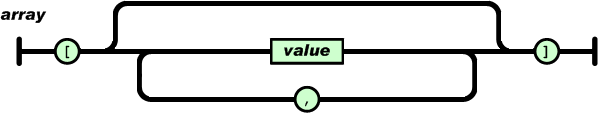


Рисунок 4.3 — Синтаксическая диаграмма массива

### 4.5.3. Значение

Значение может быть строкой в двойных кавычках, числом, true, false (т. н. boolean), null (внутренний объект JavaScript, который чаще всего интерпретируется как false), объектом или массивом. Эти структуры данных так же могут быть вложенными друг в друга (массив объектов, например). Синтаксическая диаграмма представлена на рисунке 4.4.

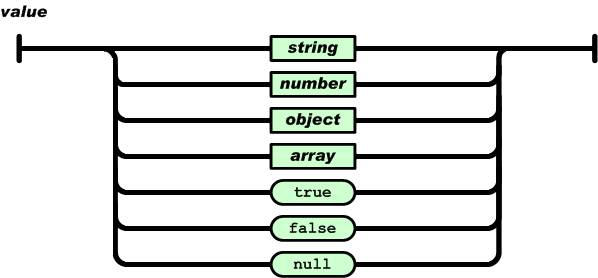


Рисунок 4.4 — Синтаксическая диаграмма значения в JSON

### 4.5.4. Обоснование выбора

Ключевым вопросом при выборе базы данных является её достаточность для текущего проекта, а также отсутствие явной избыточности (поскольку избыточный функционал влечёт за собой потери в производительности).

В рамках этого дипломного проекта предъявляются такие требования к базе данных:

* простота в использовании в сочетании с JavaScript;
* возможность сохранения базы данных в браузере;
* возможность хранения чисел и строк в упорядоченном формате, в виде вложенных структур.

JSON удовлетворяет всем этим требованиям. Он изначально был спроектирован для работы с JavaScript и позволяет сохранять объекты этого языка в том виде, в каком они представлены в программе.

Сохранение же данных реализовывается с помощью встроенного в браузер локального хранилища — localStorage. В нём можно хранить пары ключ/значения в виде строк. Поскольку JSON — это текстовый формат, его можно сохранять в виде строки, а значит такой подход вполне подходит.

### 4.5.5. Структура данных

Т. к. все данные хранятся в виде объекта, то их можно отобразить как одно дерево. Корневым узлом этого дерева будет сам объект, имеющий всего один вложенный узел — list — содержащий список созданных пользователем персональных советующих систем.

Каждая советующая система хранит в себе следующую информацию:

* уникальный номер в БД;
* название и цель;
* прецеденты;
* используемые лингвистические переменные;
* термы ситуационных векторов;
* функции принадлежности для соответствующих лингвистических термов;
* матрица знаний.

Полное дерево структуры базы данных представлено на рисунке 4.5.



Рисунок 4.5 — Структура БД в JSON

## 4.6. Проектирование и оценка подходов к реализации

При проектировании веб-приложений необходимо учитывать тот факт, что интерфейс в нём занимает большую роль, чем в приложениях для операционных систем, поскольку с ним приходится взаимодействовать напрямую. На этой почве рождаются новые подходы, отличающиеся от классического процедурного, функционального и объектно-ориентированного программирования.

### 4.6.1. MVC

Обычно при разработке веб-приложения используется классический подход под названием MVC. Model-View-Controller — это схема проектирования приложения, по которой происходит разделение пользовательского интерфейса, модели приложение и взаимодействия с пользователем на три равных, независимых друг от друга уровня, которые общаются с помощью определённого программного интерфейса [16].

Однако, в нашем случае такая схема не совсем подходит для реализации приложения.

Во-первых, приложение безсерверное и вся база данных хранится в одном объекте, к которому есть доступ из любой его точки, в силу особенностей платформы. Поэтому практически не нужны никакие вспомогательные интерфейсы для работы с базой данных.

Во-вторых, большая часть работы приложения проходит над его интерфейсом, т. к. пользователю нужно показывать большое количество форм, получать и сохранять их данные. У веб-интерфейсов есть для этого свой API, однако, если с ним работать по предложенным паттернам, понятия View и Controller сильно пересекаются друг с другом, из-за чего приложение в долгосрочной перспективе становится сложно поддерживать.

Решением в данном случае было бы использование специализированных библиотек для построения MVC, которые предлагают свои надстройки над программными интерфейсами браузера и помогают в сохранении независимости между уровнями. Однако, в таком случае приходится следовать их конструктивным особенностям и ограничениям. Другим же решением является другая методология. Например, БЭМ. Рассмотрим её подробнее.

### 4.6.2. BEM

БЭМ (BEM) — это методология, изобретённая в компании Яндекс для облегчения построения веб-приложений. Она позволяет упростить код, сделать его самодокументируемым, облегчить рефакторинг и поддержку [17]. Основная её суть в разбитии всего приложения на набор независимых друг от друга блоков (т. е. ещё большая декомпозиция, чем MVC), что, в общем-то, отражено в названии, т. к. БЭМ — это аббревиатура от «блок — элемент — модификатор» [18].

Определим основную терминологию [19]:

* **Блок** — логически и функционально независимый компонент страницы. Блок инкапсулирует в себе поведение (JavaScript), шаблоны (HTML), стили (CSS) и другие технологии реализации. Независимость блоков обеспечивает удобство в разработке и поддержке проекта.
* **Элемент** — составная часть блока, которая не может использоваться в отрыве от него.
* **Модификатор** — БЭМ-сущность, определяющая внешний вид, состояние и поведение блока или элемента.
* **БЭМ-дерево** — представление структуры веб-страницы в терминах блоков, элементов и модификаторов. Это абстракция над DOM-деревом, которая описывает имена БЭМ-сущностей, их состояния, порядок, вложенность и вспомогательные данные.

**Работа с JavaScript**

В БЭМ-методологии JavaScript используется для «оживления» веб-страницы и рассматривается как одна из технологий реализации блока.

В БЭМ к JavaScript применяются дополнительные правила, которые позволяют реализовать все идеи компонентного подхода БЭМ-методологии [20, 21].

Во-первых, весь JavaScript записывается декларативно. Декларативность JavaScript в БЭМ-проекте проявляется в следующем:

* Поведение каждого блока описывается независимо.
* Состояния блока задаются декларативно. При изменении состояний автоматически вызывается код, который задекларирован для этого состояния.
* Логика работы блока описывается как набор действий и условий, при которых эти действия необходимо выполнять. Это позволяет разделять функциональность блока на отдельные части и использовать уровни переопределения.

Во-вторых, в БЭМ-методологии к JavaScript применяются основные принципы объектно-ориентированного программирования (ООП).

* **Инкапсуляция**. В БЭМ JavaScript-реализация одного блока отделена от другого. Каждый блок предоставляет API для взаимодействия с другими блоками. Декларация блока позволяет скрыть его внутреннюю реализацию. Так как элементы всегда являются внутренней реализацией блока, обращение к ним возможно только через API самого блока.
* **Наследование**. Декларативное описание поведения блоков позволяет использовать методы базового блока внутри производного, наследовать их. Новый блок может получать все свойства и методы базового. Также можно создавать цепочки наследования — блок наследуется от другого, который, в свою очередь, наследуется от третьего и т. д.
* **Полиморфизм**. Используется на уровне прототипов, методов и свойств, поскольку JavaScript — это прототипно-ориентированный язык программирования.

Поскольку БЭМ упрощает основные моменты работы с веб-интерфейсами и облегчает разработку в будущем, а также основан на главных принципах ООП — при разработке ПО для этой работы будет использоваться именно эта методология.

## 4.7. Структура программы

### 4.7.1. Основные структурные единицы

Поскольку выбранной методологией для реализации ПО была методология БЭМ, то основными структурными единицами являются блоки. Каждый блок выполняет какую-то свою функцию, которая может быть реализована как на интерфейсном уровне, так и вовсе без него.

Список основных блоков:

* **db** — блок выполняющий действия над хранилищем localStorage;
* **controller** — абстракция над **db**, к которой имеют доступ другие блоки; позволяет выполнять основные действия с БД.
* **list** — блок отвечающий за список персональных советующих систем; позволяет создавать новые, отображает имеющиеся.
* **new-system** — блок отвечающий за создание новой советующей системы; генерирует интерфейс для ввода информации о названии и цели системы; инициализирует новую систему в базе данных.
* **settings** — блок необходимый для настройки новой советующей системы; генерирует интерфейс для ввода информации о лексических переменных и их термах; сохраняет полученную информацию в базу данных.
* **data** — блок, загружающий данные в JSON-формате о функциях принадлежности лингвистических термов и о матрице знаний.
* **thinker** —блок выполняющий основные вычисления по алгоритмам из главы 2.
* **answer —** блок производящий трансляцию результатов вычислений в формат, понятный пользователю системы и выводящий эти результаты на экран.
* **comparing** — блок отвечающий за сравнение термов лингвистических переменных, а также за указание пользователем вручную данных о функциях принадлежности.
* **rgraph** — блок инкапсулирующий в себе строение и рендеринг графиков функций принадлежности, а также операции над этими графиками.

Для общения блоков используется событийная модель. То есть, когда одному блоку нужно передать данные другому блоку, он инициирует событие, на которое подписан блок-получатель, и передаёт в него данные. Для этого используется встроенная в JavaScript событийная модель.

Также система использует следующие блоки, поставляемые с библиотеками bem-core и bem-components:

* **i-bem** — основной JavaScript-фреймворк, используемый для:
  + разработки веб-интерфейсов в терминах блоков, элементов и модификаторов;
  + описания логики работы блока в декларативном стиле — как набор состояний;
  + интегрирования JavaScript-кода с BEMHTML-шаблонами и CSS в стиле БЭМ;
  + гибкого переопределения поведения библиотечных блоков;
  + работы с DOM (инкапсулированj в элементе **dom**).
* **events** — элемент для работы с событийной моделью JavaScript; поставляет так же элемент **channel**, который предоставляет интерфейс для работы с каналами событий.
* **inherit** — предоставляет функцию, реализующую механизмы для объявления и наследования классов (абстракции классов изначально нет в JavaScript).
* **jquery** — реализация библиотеки jQuery в технологиях БЭМ.

### 4.7.2. Зависимости структурных единиц

Зависимости блоков, описанных в п. 4.7.1, представлены в виде UML-диаграммы на рисунке 4.6.

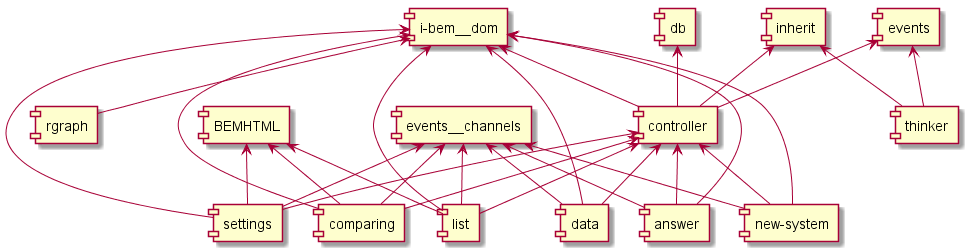


Рисунок 4.6 — UML-диаграмма зависимостей блоков

А на рисунке 4.7 представлена UML-диаграмма описывающая основные методы, поля блоков (значок «B») и элементов (значок «E»).

## 4.8. Работа программы

На главном экране программы отображается список уже созданных ситуационных советующих систем и кнопка для добавления новой (рисунок 4.8).

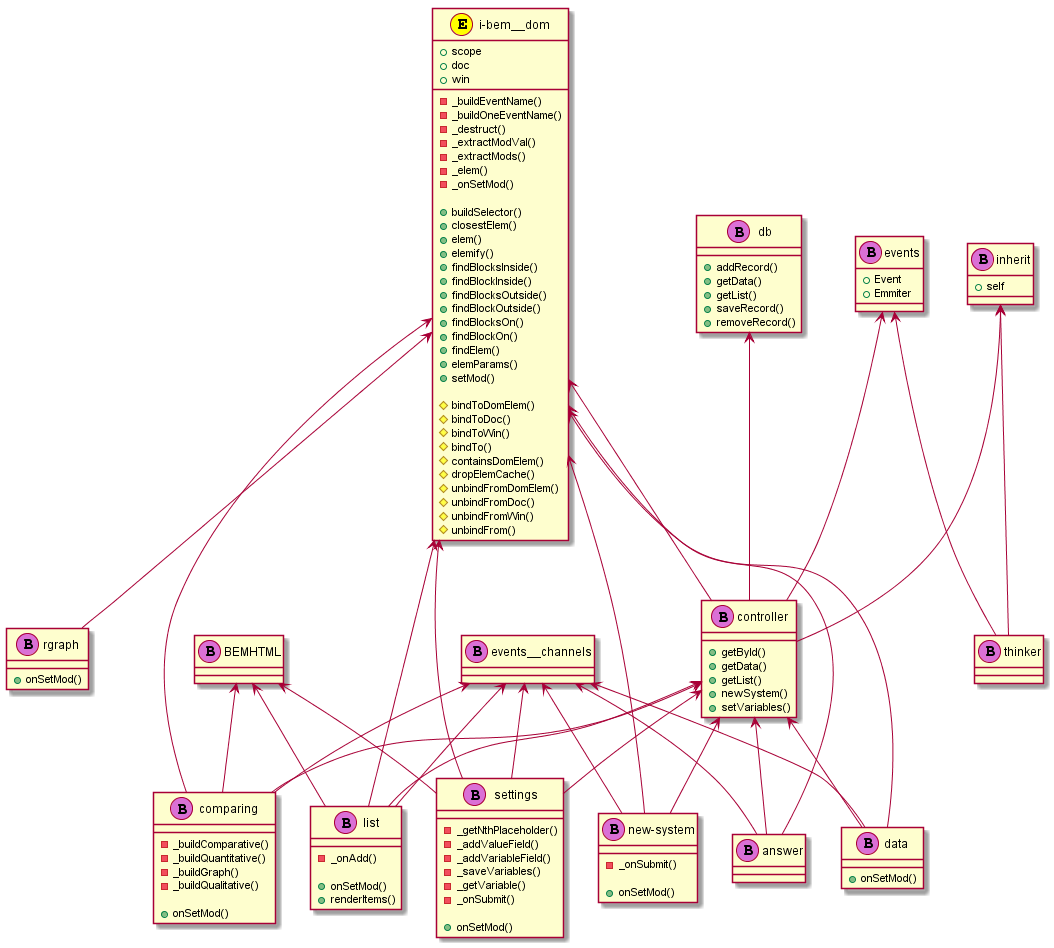


Рисунок 4.7 — UML-диаграмма методов и полей блоков и элементов

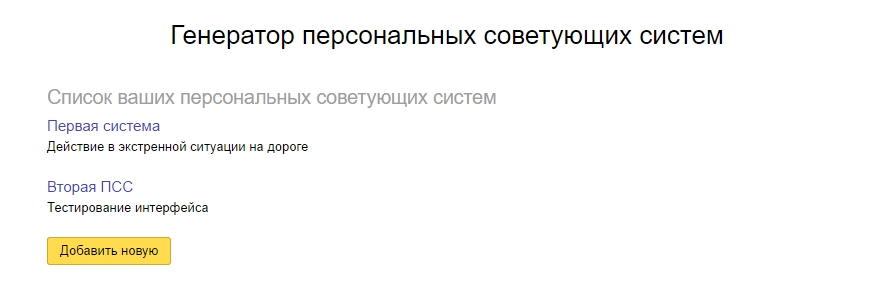


Рисунок 4.8 — Главная страница

После нажатия на кнопку добавления пользователь попадает на экран ввода начальной информации о ППС (рисунок 4.9).



Рисунок 4.9 — Ввод названия и цели ПСС

Во всём интерфейсе программы предусмотрены обязательные поля, без которых пользователь не сможет перейти на другой шаг. Так, если он не ввёл название для ситуационной советующей системы, программа его об этом предупредит (рисунок 4.10).

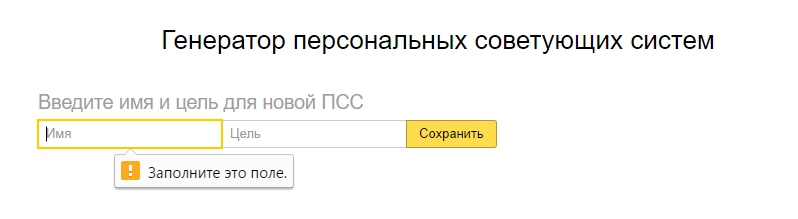


Рисунок 4.10 — Уведомление об ошибке при заполнении формы

После заполнения названия и цели пользователь переходит ко вводу лингвистических переменных и их термов (рисунок 4.11).

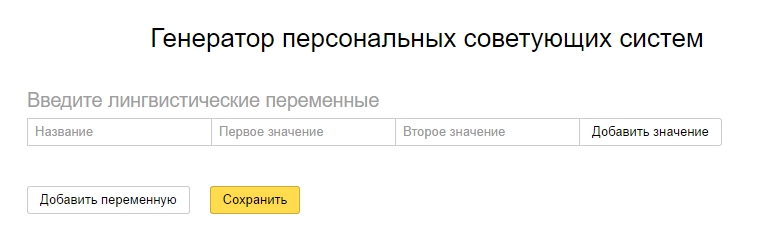


Рисунок 4.11 — Ввод информации о лингвистических переменных

Следующим шагом будет ввод информации о том, какой из термов является качественным, а какой количественным (рисунок 4.12).

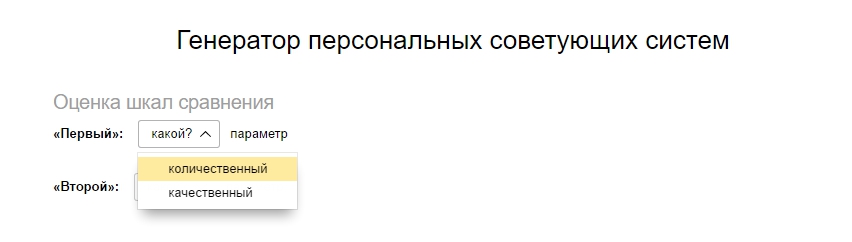


Рисунок 4.12 — Указание типов параметров

Ввод информации об универсальном множестве (рисунок 4.13).

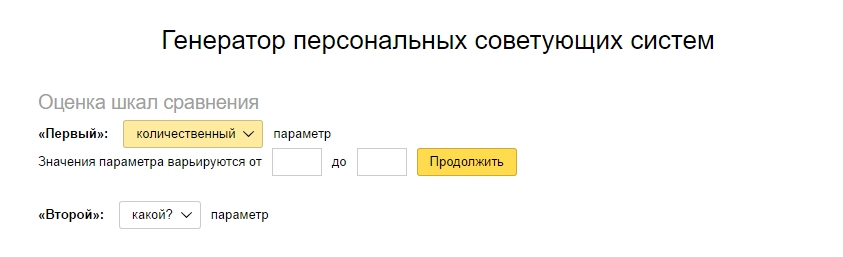


Рисунок 4.13 — Указание информации об универсальном множестве

А также ввод матриц принадлежностей (рисунок 4.14).



Рисунок 4.14 — Ввод информации о матрице принадлежности

После чего пользователь загружает информацию о матрице знаний, информацию о текущей ситуации. И получает ответ от программы о том, какой прецедент в данном контексте является наилучшим.

## 4.9. Тестирование программы

В качестве тестирования работоспособности программы сгенерируем с её помощью советующую систему для задачи, описанной в [11].

Итак, решается проблема выбора тактики поведения пилота на основании ниже следующих данных:

* Ситуационный вектор:
* Лингвистические переменные и их термы:
  + «Дальность до цели» = {«Очень малая», «Малая», «Средняя», «Большая», «Очень большая»};
  + «Видимость» = {«Нулевая», «Малая», «Средняя», «Большая», «Отличная»};
  + «Высота» = {«Минимальная», «Малая», «Средняя», «Большая», «Максимальная»};
  + «Скорость» = {«Минимальная», «Малая», «Средняя», «Большая», «Максимальная»};
  + «Количество горючего» = {«Почти нет», «Мало», «Средне», «Достаточное», «Полные баки»};
  + «Количество боеприпасов» = {«Почти нет», «Мало», «Средне», «Достаточное», «Полный боезапас»};
  + «Мастерство лётчика» = {«Очень низкое», «Низкое», «Среднее», «Высокое», «Очень высокое»};
  + «Состояние летательного аппарата» = {«Еле летит», «Пока летит», «Без критических повреждений», «Хорошее», «Очень хорошее»};
  + «Метеорологические условия» = {«Очень плохие», «Плохие», «Нормальные», «Хорошие», «Отличные»};
  + «Ветер» = {«Почти нет», «Слабый», «Средний», «Сильный», «Очень сильный»}.
* Функции принадлежности для соответствующих лингвистических термов (задаются пользователем).
* Прецеденты (исходы): «Сесть на ближайший аэродром», «Возвращаться на базу», «Продолжить, быть готовым вернуться», «Продолжить, соблюдать осторожность», «Продолжить задание в штатном режиме».
* Заполненная матрица знаний в виде массива. Строка данной матрицы соответствует продукционному правилу, столбец — координате ситуационного вектора, число — номер терма соответствующей координаты ситуационного вектора (нумерация термов начинается с 0):

[{0, 0, 4, 4, 4, 4, 4, 4, 1, 4},

{4, 1, 1, 1, 2, 2, 4, 0, 0, 1},

{2, 2, 2, 4, 2, 1, 0, 1, 2, 4},

{2, 2, 4, 1, 2, 4, 4, 1, 2, 1},

{4, 4, 4, 2, 4, 2, 4, 1, 4, 2},

{4, 4, 4, 4, 1, 4, 4, 1, 0, 4},

{4, 4, 0, 0, 0, 4, 4, 1, 2, 2},

{4, 4, 4, 0, 4, 4, 4, 2, 4, 0},

{4, 4, 4, 2, 0, 4, 4, 0, 1, 0},

{0, 4, 4, 0, 1, 2, 4, 4, 2, 4},

{4, 1, 0, 0, 0, 4, 4, 1, 1, 1}]

Имея эти данные, пользователь заходит на страницу генератора персональных советующих систем и нажимает на кнопку создания новой системы (рисунок 4.15).

Затем вводит название создаваемой советующей системы (например, «Авиация») и цель (рисунок 4.16).

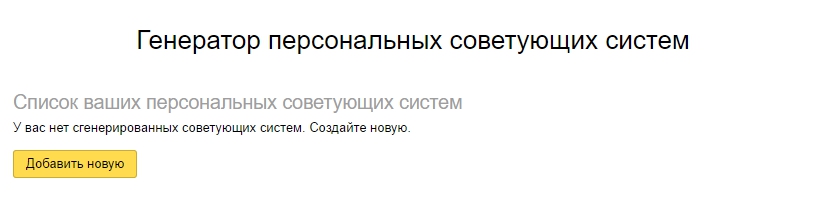


Рисунок 4.15 — Главная страница ГПСС

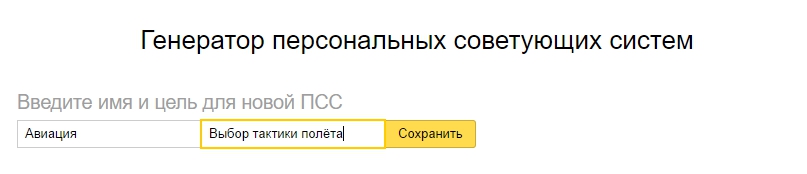


Рисунок 4.16 — Ввод имени и цели ПСС

После ввода имени и цели пользователь оказывается на экране ввода лингвистических переменных и их термов. Заполненная форма с ними представлена на рисунке 4.17.

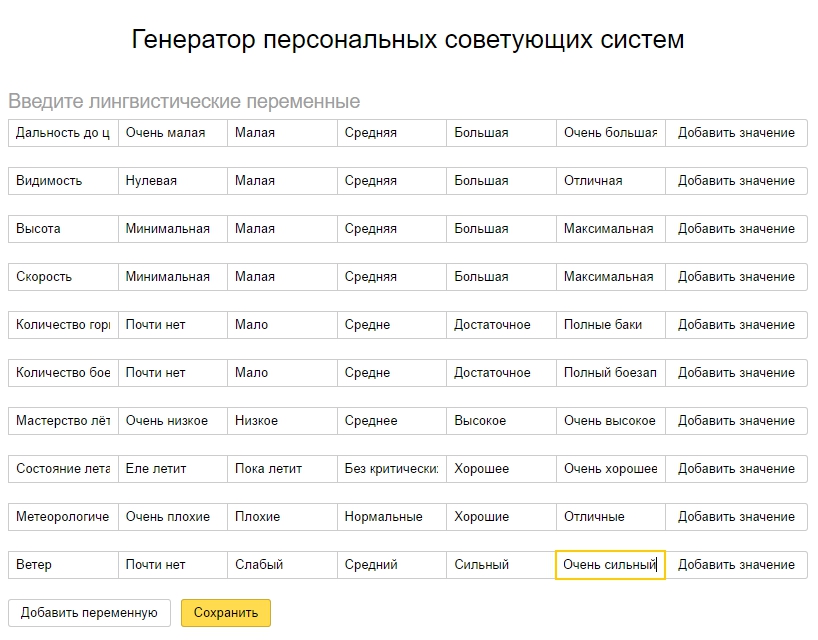


Рисунок 4.17 — Ввод лингвистических переменных

Когда пользователь заполнил форму с лингвистическими переменными, ему предоставляется возможность оценки шкал сравнения. То есть, для тех лингвистических переменных, которые имеют количественное выражение (в нашем случае это, например, высота) необходимо задать функцию принадлежности, а для количественных необходимо сравнить друг с другом термы с помощью матрицы парных сравнений. Незаполненная форма показана на рисунке 4.18.

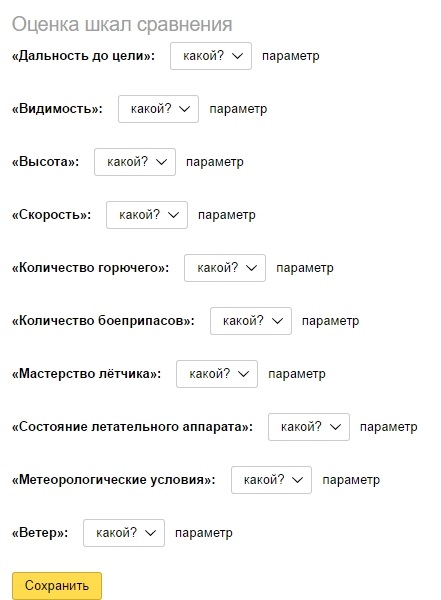


Рисунок 4.18 — Незаполненная форма оценки шкал сравнения

Форма, на которой заданы универсальные множества — на рисунке 4.19. А на рисунке 4.20 видна форма с заданными функциями принадлежностей.

После указания функций принадлежности, необходимо загрузить данные о матрице знаний. На рисунке 4.21 показана форма для загрузки файла с ними.

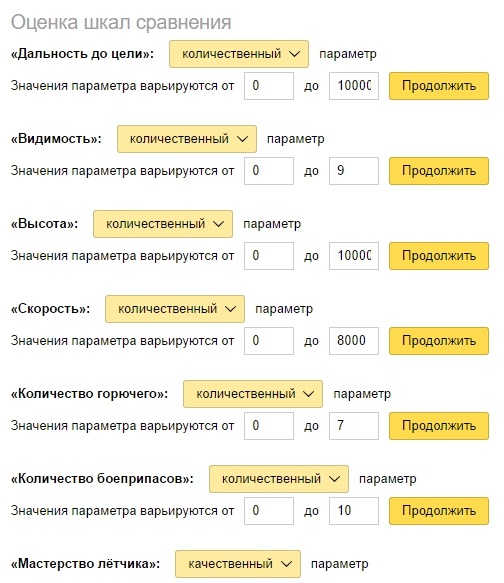


Рисунок 4.19 — Форма оценки шкал сравнения с заданными универсальными множествами

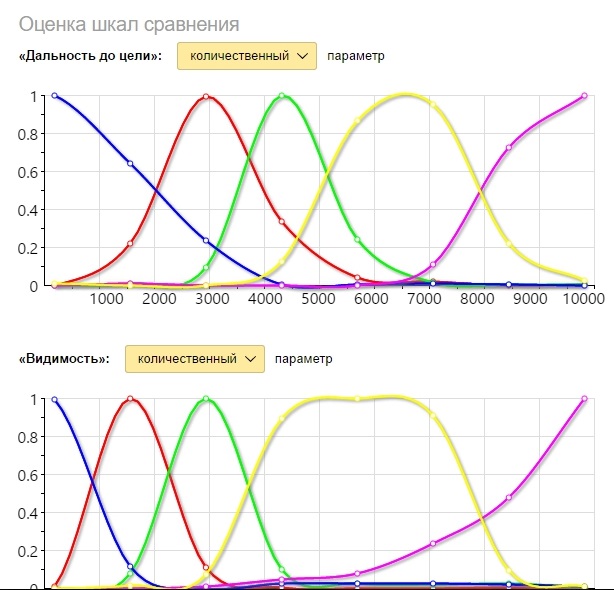


Рисунок 4.20 — Функции принадлежности



Рисунок 4.21 — Форма загрузки файла с матрицей знаний

Последним шагом является заполнение формы с информацией о текущей ситуации. В заполненном виде она показана на рисунке 4.22.

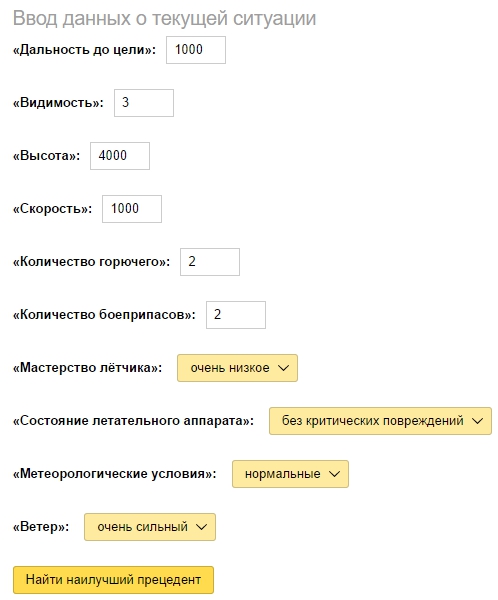


Рисунок 4.22 — Форма ввода данных о текущей ситуации

После нажатия на кнопку «Найти наилучший прецедент» пользователь получает ответ от программы (рисунок 4.23).

После нахождения наилучшего прецедента сгенерированная советующая система доступна для повторного использования без необходимости введения информации о функциях принадлежности и пр. (рисунок 4.24).

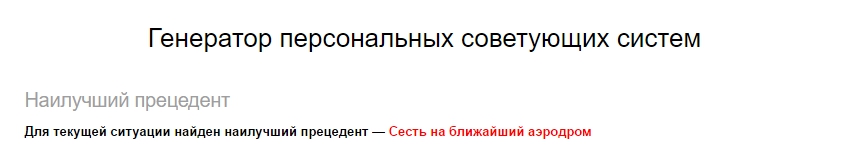


Рисунок 4.23 — Наилучший прецедент

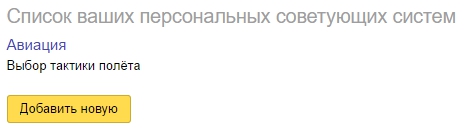


Рисунок 4.24 — Сгенерированная советующая система

Вывод

В этом разделе были рассмотрены инструменты, методологии и подходы, использовавшиеся для создания программного обеспечения для генерации персональных ситуационных советующих систем. В результате получилось веб-приложение, которое хранит историю обо всех советующих системах пользователя и позволяет в любой момент получить наилучший прецедент по информации о текущей ситуации.

# Заключение

В ходе выполнения данной дипломной работы были рассмотрены основные задачи ситуационных советующих систем, алгоритмы их построения и доказана актуальность их использования. В качестве упрощения процесса построения персональных советующих систем был предложен и разработан генератор персональных ситуационных советующих систем.

Для создания ПО были выбраны наиболее актуальные на данный момент инструментальные средства, обеспечивающие быструю и надёжную разработку, а также беспроблемную поддержку в будущем.

Так как разработанное ПО является веб-приложением, было предложено отказаться от базы данных на отдельном сервере и хранить все данные на стороне клиента. Эффективность и польза такого подхода была доказана, выбор инструментария обоснован.

Для тестирования работы системы был выбран набор реальных данных. В ходе тестирования был найден оптимальный прецедент для проблемной ситуации. Таким образом можно утверждать, что ПО удовлетворяет требованиям технического задания и упрощает задачу анализа данных эксперту.

# Список литературы

1. Поспелов Д. А. Логико-лингвистические модели в системах управления. М.: Энергоиздат, 1981. 231 с.
2. Мелихов А. Н., Берштейн Л. С., Коровин С. Я. Ситуационные советующие системы с нечёткой логикой. М.: Наука. Гл. ред. физ.-мат. лит., 1990. 272 с.
3. Пушкин В. Н. Оперативное мышление в больших системах. М.: Энергия, 1965. 257 с.
4. Тихомиров О. К. Структура мыслительной деятельности человека. М.: Изд-во МГУ, 1967. 158 с.
5. Заде Л. Понятие лингвистической переменной и его применение к принятию приближённых решений. М.: Мир, 1976. 165 с.
6. Кофман А. Введение в теорию нечётких множеств. М.: Радио и связь, 1982. 432 с.
7. Аверкин А. Н., Батыршин И. З., Блишун А. Ф. Нечёткие множества в моделях управления и искусственного интеллекта / Под ред. Поспелова Д. А. М.: Наука, 1986. 312 с.
8. Борисов А. Н., Алексеев А. В., Крумберг О. А. Модели принятия решений на основе лингвистической переменной. Рига: Зинатне, 1982. 256 с.
9. О языке программирования JavaScript. [Электронный ресурс] URL: <http://igoradamenko.com/bstu/1> (дата обращения: 05.06.2016).
10. Ротштейн А. П. Интеллектуальные технологии идентификации. Винница: Универсум, 1999. 295 с.
11. Федунов Б. Е., Прохоров М. Д. Вывод по прецеденту в базах знаний бортовых интеллектуальных систем // Искусственный интеллект и принятие решений 3/2010, с. 63–72.
12. Федунов Б. Е. Механизмы вывода в базе знаний бортовых оперативно советующих экспертных систем. // Изв. РАН. ТиСУ. 2002. №4
13. Варшавский П. Р., Еремеев А. П. Моделирование рассуждений на основе прецедентов в интеллектуальных системах поддержки принятия решений. // «Искусственный интеллект и принятие решений». №2. 2009. с. 45–47.
14. Прохоренко А. М., Никитенко А. А., Ерёменко Д. С. Применение ситуационных моделей в системах управления технологическими процессами [Электронный ресурс] URL: <http://igoradamenko.com/bstu/2> (дата обращения: 07.06.2016).
15. О редакторе кода Sublime Text [Электронный ресурс] URL: <http://igoradamenko.com/bstu/3> (дата обращения: 08.06.2016).
16. Шаблон проектирования Model-View-Controller [Электронный ресурс] URL: <http://igoradamenko.com/bstu/4> (дата обращения: 10.06.2016).
17. История создания / Методология / БЭМ [Электронный ресурс] URL: <http://igoradamenko.com/bstu/5> (дата обращения: 11.06.2016).
18. Какие проблемы решает / Методология / БЭМ [Электронный ресурс] URL: <http://igoradamenko.com/bstu/6> (дата обращения: 11.06.2016).
19. Основные понятия / Методология / БЭМ [Электронный ресурс] URL: <http://igoradamenko.com/bstu/7> (дата обращения: 11.06.2016).
20. JavaScript / Методология / БЭМ [Электронный ресурс] URL: <http://igoradamenko.com/bstu/8> (дата обращения: 11.06.2016).
21. Особенности JavaScript / Методология / БЭМ [Электронный ресурс] URL: <http://igoradamenko.com/bstu/9> (дата обращения: 12.06.2016).

# Приложение А

**Техническое задание**

А.1. Общие сведения

А.1.1. Полное наименование и условное обозначение системы

«Генератор персональных ситуационных советующих систем»

А.1.2. Наименование заказчика и исполнителя, их реквизиты

Наименование Заказчика

|  |  |
| --- | --- |
| *Заказчик:* | Федеральное государственное бюджетное образовательное учреждение высшего образования «Белгородский государственный технологический университет им. В. Г. Шухова» |
| *Юридический адрес:* | Россия, 308012, г. Белгород, ул. Костюкова 46, БГТУ им. В. Г. Шухова |
| *Телефон:* | +7 (4722) 54-98-53 |
| *Контактное лицо:* | Поляков В. М. |

Наименование Исполнителя

|  |  |
| --- | --- |
| *Исполнитель:* | Адаменко И. И. |
| *Телефон:* | +7 905 173-89-94 |
| *E-mail:* | [mail@igoradamenko.com](mailto:mail@igoradamenko.com) |

А.1.3. Перечень документов, на основании которых создаётся система

Система разрабатывается в рамках выпускной квалификационной работы на основании учебного плана направления «231000.62 — Программная инже-

нерия».

А.1.4. Плановые сроки начала и окончания работ

Начало работ: сентябрь 2015 г.

Конец работ: май 2016 г.

А.1.5. Порядок оформления и предъявления результатов работ

Система передается в виде функционирующего комплекса на базе средств вычислительной техники Заказчика и Исполнителя в установленные сроки. Приемка системы осуществляется комиссией в составе уполномоченных представителей Заказчика.

А.2. Назначение и цели создания системы

А.2.1. Назначение системы

Разрабатываемый программный продукт предназначен для генерации систем для моделирования проблемной ситуации и оперативного вывода по прецеденту. Задавая исходные данные как с помощью количественных, так и с качественных оценок, пользователь может воспроизвести текущую ситуацию и получить совет о наилучшем выходе из неё, основываясь на предыдущих прецедентах.

А.2.2. Цели и задачи создания системы

Основными целями являются:

* моделирование проблемной ситуации;
* нахождение оптимального прецедента в текущей проблемной ситуации.

А.3. Требования к системе

А.3.1. Требования к функциональным характеристикам

Программный продукт должен обеспечивать успешное выполнение следующих функций:

1. ввод количественных и качественных характеристик текущей ситуации;
2. перевод введённых данных на язык нечётких множеств;
3. ввод данных об успешных прецедентах из файла;
4. определение предпочтительного прецедента для разрешения конкретной проблемной ситуации;
5. сохранение результатов вычислений с последующим доступом к ним.

Входные данные от пользователей:

1. название и цель ситуационной советующей системы;
2. количественные и качественные характеристики ситуации;
3. данные о функциях принадлежности лингвистических термов;
4. файл с базой данных об успешных прецедентах в формате JSON.

Выходные данные программного продукта:

1. файл с примером структуры JSON-файла для базы данных об успешных прецедентах;
2. информация о предпочтительном прецеденте в текущей ситуации;
3. изменённая база данных успешных прецедентов.

А.3.2. Требования к численности и квалификации персонала

Для эксплуатации ПО достаточно одного человека — эксперта. Он генерирует ситуационную систему по имеющимся у него данным и затем использует её в проблемных ситуациях подобного класса.

Возможно использование ПО более, чем одним человеком. Необходим как минимум один эксперт для генерации ситуационной системы, пользоваться же ей могут другие пользователи.

А.3.3. Требования к надёжности

Для обеспечения надёжной и безошибочной работы программы, не обходимо предусмотреть контроль вводимых данных и оповещения пользователя о неправильных действиях при работе с ней.

А.3.4. Требования к безопасности

Безопасность должна обеспечиваться следующими требованиями:

* соблюдение требований по безопасности средств вычислительной техники;
* соблюдение условий эксплуатации ЭВМ в помещениях;
* соблюдение требований пожарной безопасности.

А.3.5. Требования к составу и параметрам технических средств

Разрабатываемое программное обеспечение должно работать без ошибок на IBM-совместимых компьютерах со следующей минимальной конфигурацией:

1. процессор Inter Pentium 4 с тактовой частотой 1 ГГц, или аналогичный ему (SSE2-совместимый) AMD с тактовой частотой 1,5 ГГц;
2. оперативная память от 128 Мб;
3. запоминающее устройство со свободным объёмом памяти 128 Мб;
4. клавиатура;
5. монитор;
6. операционная система Windows XP / Ubuntu 8.04 / OS X Mavericks 10.9 / Android 4.0 / iOS 7, или любые другие совместимые с ними (в зависимости от аппаратной платформы).

А.3.6. Требования к программной документации

Программные модули должны быть разработаны таким образом, чтобы быть самодокументируемыми. Интерфейс программы должен быть интуитивно понятен пользователю, а неявные моменты пояснены включённой в ПО справочной документацией.

А.4. Порядок контроля и приёмки

Должны выполняться основные функции, заявленные в техническом задании.

Испытание технических и программных средств системы должно осуществляться путем их тестирования в целях выявления возможных ошибок и недостатков. Испытания по приёму и контролю системы должны проводиться на объекте Заказчика в условленные сроки.

Ответственность за организацию и проведение проверки системы должен нести заказчик.

# Приложение Б

**Исходный код приложения**

Б.1. Блок comparing

1. modules.define('comparing', ['BEMHTML', 'i-bem\_\_dom',  
   'events\_\_channels', 'controller'], **function**(provide, BEMHTML, BEMDOM,  
   channels, Controller) {
3. provide(BEMDOM.decl(**this**.name, {
4. onSetMod: {
5. js: {
6. inited: **function**() {
7. **var** that = **this**;
9. channels('events').on('comparing', **function**(e, id) {
10. that.setMod('visible');
11. that.id = id;
12. that.variables = (Controller.getById(id)).lingualVars;
14. that.\_buildComparative();
15. });
17. channels('events').emit('comparing', 1);
18. }
19. }
20. },
22. \_buildComparative: **function**() {
23. **var** vars = **this**.variables,
24. that = **this**;
26. **for** (**var** i = 0; i < vars.length; i++) {
27. BEMDOM.append(
28. **this**.domElem,
29. BEMHTML.apply({
30. block: 'row',
31. mix: { block: 'comparing', elem: 'line', mods: { num: i }
32. },
33. content: {
34. elem: 'col',
35. content: [
36. {
37. block: 'row',

1. content: [
2. {
3. elem: 'col',
5. tag: 'strong',
6. mix: { block: 'comparing', elem: 'tip' },
7. content: '«' + vars[i].name + '»:'
8. },
9. {
10. elem: 'col',
11. content: {
12. block: 'select',
13. mods: { mode: 'radio-check', theme: 'islands', size: 'm' },
14. mix: { block: 'comparing', elem: 'var-type' },
15. name: 'select' + i,
16. text: 'какой?',
17. options: [
18. { val: 'quantitative', text:  
    'количественный' },
19. { val: 'qualitative', text:  
    'качественный' }
20. ]
21. }
22. },
23. {
24. elem: 'col',
25. mix: { block: 'comparing', elem: 'tip' },
26. content: 'параметр'
27. }
28. ]
29. },
30. {
31. block: 'row',
32. mix: { block: 'comparing', elem: 'var-type-data' },
33. content: ''
34. }
35. ]
36. }
37. })
38. );
39. }
41. **var** selects = **this**.findBlocksInside('var-type', 'select');
43. **for** (**var** i = selects.length - 1; i >= 0; i--) {
44. selects[i].on('change', **function**(e) {
45. **var** $target = **this**.domElem.closest('.row').next(),
46. id = **this**.domElem.data('bem').select.name.replace(  
    'select', '');
48. **if** (**this**.getVal() == 'quantitative') {
49. that.\_buildQuantitative($target, id);
50. } **else** {
51. that.\_buildQualitative($target, id);
52. }
53. });
54. }
55. },
57. \_buildQuantitative: **function**(target, id) {
58. BEMDOM.update(
59. target,
60. BEMHTML.apply([
61. {
62. block: 'row',
63. elem: 'col',
64. mix: { block: 'comparing', elem: 'tip' },
65. content: 'Значения параметра варьируются от'
66. },
67. {
68. block: 'input',
69. mods: { theme: 'islands', size: 'm'},
70. mix: { block: 'comparing', elem: 'range', mods: { type: 'min' } },
71. required: 'required'
72. },
73. {
74. block: 'row',
75. elem: 'col',
76. mix: { block: 'comparing', elem: 'tip' },
77. content: 'до'
78. },
79. {
80. block: 'input',
81. mods: { theme: 'islands', size: 'm'},
82. mix: { block: 'comparing', elem: 'range', mods: { type: 'max' } },
83. required: 'required'
84. },
85. {
86. block: 'button',
87. mods: { theme: 'islands', size: 'm', view: 'action' },
88. mix: { block: 'comparing', elem: 'quantitative-next' },
89. attrs: {
90. 'data-id': id
91. },
92. text: 'Продолжить'
93. }
94. ])
95. );
97. **this**.unbindFrom('quantitative-next', 'click');
99. **this**.bindTo('quantitative-next', 'click', **function**(e) {
100. **var** vals = **this**.findBlocksInside(target, 'input').map(  
     **function**(item, index) {
101. **return** item.getVal();
102. }),
103. id = $(e.target).closest('.button').data('id');
105. **this**.\_buildGraph(target, vals, id);
106. });
108. **this**.dropElemCache('quantitative-next');
109. },
111. \_buildGraph: **function**(target, range, id) {
112. BEMDOM.update(
113. target,
114. BEMHTML.apply({
115. block: 'row',
116. elem: 'col',
117. content: {
118. block: 'row',
119. content: {
120. block: 'rgraph',
121. js: **true**,
122. mix: { block: 'comparing', elem: 'graph' },
123. attrs: {
124. uid: id,
125. min: **new** Number(range[0]),
126. max: **new** Number(range[1]),
127. data: [[.4, .5, .8, .7, .6, .4, .3, .5], [.7, .1, .6, .9, .4, .6, .5, .2]]
128. }
129. }
130. }
131. })
132. );
133. },
135. \_buildQualitative: **function**(target) {
136. BEMDOM.update(
137. target,
138. BEMHTML.apply({
139. block: 'row',
140. elem: 'col',
141. content: 'кол'
142. })
143. );
144. }
145. }));
147. })

Б.2. Блок controller

1. modules.define('controller', ['i-bem\_\_dom', 'inherit', 'events', 'db'],  
   **function**(provide, BEMDOM, inherit, events, DB) {
3. **var** Controller = inherit(events.Emitter, {
4. \_\_constructor: **function**() {
5. },
7. setVariables: **function**(id, variables) {
8. **var** rec = DB.getRecord(id);
10. rec.lingualVars = variables;
12. **return** DB.saveRecord(id, rec);
13. },
15. getList: **function**() {
16. **return** DB.getList();
17. },
19. getData: **function**() {
20. **return** DB.getData();
21. },
23. getById: **function**(id) {
24. **return** **this**.getData().list[id - 1];
25. },
27. newSystem: **function**(name, goal) {
28. **return** DB.addRecord({
29. name: name,
30. goal: goal
31. });
32. }
33. });
35. provide(**new** Controller());
37. })

Б.3. Блок db

1. modules.define('db', **function**(provide) {
3. **var** data = JSON.parse(localStorage.getItem('data')) || {
4. list: []
5. };
7. **function** save() {
8. localStorage.setItem('data', JSON.stringify(data));
9. }
11. provide({
12. getList: **function**() {
13. **return** data.list.map(**function**(item, index) {
14. **return** {
15. name: item.name,
16. goal: item.goal,
17. id: item.id
18. };
19. });
20. },
22. addRecord: **function**(rec) {
23. data.list.push(rec);
25. save();
27. **return** data.list.length;
28. },
30. saveRecord: **function**(id, rec) {
31. data.list[id - 1] = rec;
33. save();
34. },
36. getRecord: **function**(id) {
37. **return** data.list[id - 1];
38. },
40. getData: **function**(id) {
41. **return** data;
42. }
43. });
45. })

Б.4. Блок list

Клиентская часть:

1. modules.define('list', ['BEMHTML', 'i-bem\_\_dom', 'events\_\_channels',  
   'controller'], **function**(provide, BEMHTML, BEMDOM, channels, Controller) {
3. provide(BEMDOM.decl(**this**.name, {
4. onSetMod: {
5. js: {
6. inited: **function**() {
7. **var** list = Controller.getList();
9. **this**.renderItems(list);
11. **this**.bindTo('add', 'click', **function**() {
12. **this**.\_onAdd();
13. });
14. }
15. }
16. },
18. \_onAdd: **function**() {
19. channels('events').emit('new-system');
21. **this**.delMod('visible');
22. },
24. renderItems: **function**(list) {
25. BEMDOM.append(
26. **this**.elem('content'),
27. BEMHTML.apply(list.map(**function**(item, index) {
28. **return** {
29. block: 'list',
30. elem: 'item',
31. id: item.id,
32. desc: item.goal,
33. content: item.name
34. }
35. }))
36. );
37. }
38. }));
40. })

Шаблон:

1. block('list').elem('item')(
2. content()(**function**() {
3. **var** ctx = **this**.ctx;
5. **return** [
6. {
7. block: 'link',
8. mods: {
9. pseudo: **true**,
10. theme: 'islands',
11. size: 'l'
12. },
13. url: '#' + ctx.id,
14. title: ctx.content,
15. content: ctx.content
16. },
17. {
18. elem: 'desc',
19. content: ctx.desc
20. }
21. ];
22. })
23. );

Б.5. Блок new-system

1. modules.define('new-system', ['i-bem\_\_dom', 'events\_\_channels',  
   'controller'], **function**(provide, BEMDOM, channels, Controller) {
3. provide(BEMDOM.decl(**this**.name, {
4. onSetMod: {
5. js: {
6. inited: **function**() {
7. **var** that = **this**;
9. channels('events').on('new-system', **function**() {
10. that.setMod('visible');
11. });
13. **this**.bindTo('form', 'submit', **function**(e) {
14. e.preventDefault();
16. **this**.\_onSubmit();
17. });
18. }
19. }
20. },
22. \_onSubmit: **function**() {
23. **var** id = Controller.newSystem(
24. **this**.findBlockOn('name', 'input').getVal(),
25. **this**.findBlockOn('goal', 'input').getVal()
26. );
28. channels('events').emit('settings', id);
30. **this**.delMod('visible');
31. }
32. }));
34. })

Б.6. Блок rgraph

Клиентская часть:

1. modules.define('rgraph', ['i-bem\_\_dom'], **function**(provide, BEMDOM) {
3. **function** numFormatter(obj, num) {
4. **return** **new** Number(num).toString();
5. }
7. **var** settings = {
8. options: {
9. gutterBottom: 35,
10. linewidth: 2,
11. hmargin: 10,
12. shadow: **true**,
13. adjustable: **true**,
14. spline: **true**,
15. tickmarks: 'circle',
16. ticksize: 2,
17. textAccessible: **true**,
18. scaleDecimals: 2,
19. ymin: 0,
20. ymax: 1,
21. scaleFormatter: numFormatter
22. }
23. };
25. provide(BEMDOM.decl(**this**.name, {
26. onSetMod: {
27. js: {
28. inited: **function**() {
29. **var** options = settings,
30. min = $(**this**.domElem).data('min'),
31. max = $(**this**.domElem).data('max');
33. **var** labels = [min],
34. d = (max - min) / 10,
35. cur = min;
37. **for** (**var** i = 1; i < 10; i++) {
38. cur += d;
39. labels.push(+(cur).toFixed(2));
40. }
42. labels.push(max);
44. options.id = 'rgraph' + JSON.parse($(**this**.domElem).data('uid'));
45. options.data = $(**this**.domElem).data('data');
46. options.options.labels = labels;
48. **var** line = **new** RGraph.Line(options).draw();
50. window[options.id] = line;
51. }
52. }
53. }
54. }));
56. })

Шаблон:

1. block('rgraph')(
2. tag()('canvas'),
3. attrs()(**function**() {
4. **var** ctx = **this**.ctx,
5. attrs = {
6. id: 'rgraph' + ctx.attrs.uid,
7. width: 600,
8. height: 250
9. };
11. **for** (attr **in** ctx.attrs) {
12. attrs['data-' + attr] = JSON.stringify(ctx.attrs[attr]);
13. }
15. **return** attrs;
16. })
17. );

Б.7. Блок settings

1. modules.define('settings', ['BEMHTML', 'i-bem\_\_dom',  
   'events\_\_channels', 'jquery', 'controller'], **function**(  
   provide, BEMHTML, BEMDOM, channels, $, Controller) {
3. **var** placeholders = [
4. 'Первое',
5. 'Второе',
6. 'Третье',
7. 'Четвёртое',
8. 'Пятое',
9. 'Шестое',
10. 'Седьмое',
11. 'Восьмое',
12. 'Девятое',
13. 'Десятое',
14. 'Ещё одно'
15. ].map(**function**(item) {
16. **return** item + ' значение';
17. });
19. provide(BEMDOM.decl(**this**.name, {
20. onSetMod: {
21. js: {
22. inited: **function**() {
23. **var** that = **this**;
25. channels('events').on('settings', **function**(e, id) {
26. that.setMod('visible');
27. that.id = id;
28. });
30. **this**.bindTo('add-value', 'click', **function**(e) {
31. **this**.\_addValueField(**this**.closestElem($(e.target),  
    'add-value'));
32. });
34. **this**.bindTo('add-variable', 'click', **function**(e) {
35. **this**.\_addVariableField(**this**.closestElem($(e.target), 'last-row'));
36. });
38. **this**.bindToDomElem(**this**.domElem, 'submit', **function**(e) {
39. e.preventDefault();
41. **this**.\_onSubmit();
42. });
43. }
44. }
45. },
47. \_getNthPlaceholder: **function**(n) {
48. **return** placeholders[n > 10 ? 10 : n - 1];
49. },
51. \_addValueField: **function**(target) {
52. BEMDOM.before(
53. target,
54. BEMHTML.apply({
55. block: 'input',
56. mods: { theme: 'islands', size: 'm' },
57. mix: { block: 'settings', elem: 'value' },
58. required: 'required',
59. placeholder: **this**.\_getNthPlaceholder($(target).siblings(  
    **this**.buildSelector('value')).length + 1)
60. })
61. );
62. },
64. \_addVariableField: **function**(target) {
65. BEMDOM.before(
66. target,
67. BEMHTML.apply({
68. block: 'row',
69. content: [
70. {
71. elem: 'col',
72. mods: {
73. m: **true**
74. },
75. content: {
76. block: 'control-group',
77. mix: { block: 'settings', elem: 'ling-var' },
78. content: [
79. {
80. block: 'input',
81. mix: { block: 'settings', elem: 'name' },
82. mods: { theme: 'islands', size: 'm' },
83. required: 'required',
84. placeholder: 'Название'
85. },
86. {
87. block: 'input',
88. mix: { block: 'settings', elem:   
    'value' },
89. mods: { theme: 'islands', size: 'm' },
90. required: 'required',
91. placeholder: 'Первое значение'
92. },
93. {
94. block: 'input',
95. mods: { theme: 'islands', size: 'm' },
96. mix: { block: 'settings', elem:   
    'value' },
97. required: 'required',
98. placeholder: 'Второе значение'
99. },
100. {
101. block: 'button',
102. mods: { theme: 'islands', size: 'm' },
103. mix: { block: 'settings', elem:   
     'add-value' },
104. text: 'Добавить значение'
105. }
106. ]
107. }
108. }
109. ]
110. })
111. );
113. **this**.dropElemCache('add-value');
115. **this**.unbindFrom('add-value', 'click');
117. **this**.bindTo('add-value', 'click', **function**(e) {
118. **this**.\_addValueField(**this**.closestElem(e.target, 'add-value'));
119. });
120. },
122. \_saveVariables: **function**() {
123. Controller.setVariables(
124. **this**.id,
125. **this**.findElem('ling-var')
126. .map(**this**.\_getVariable.bind(**this**))
127. .get()
128. );
129. },
131. \_getVariable: **function**(index, node) {
132. **var** that = **this**,
133. name = **this**.findElem($(node), 'name')
134. .map(**function**(i, item) {
135. **return** that.findBlockOn($(item), 'input').getVal();
136. })
137. .get(),
138. values = **this**.findElem($(node), 'value')
139. .map(**function**(i, item) {
140. **return** that.findBlockOn($(item), 'input').getVal();
141. })
142. .get();
144. **return** {
145. name: name,
146. values: values
147. };
148. },
150. \_onSubmit: **function**() {
151. **this**.\_saveVariables();
153. channels('events').emit('comparing', **this**.id);
155. **this**.delMod('visible');
156. }
157. }));
159. })

Б.8. Основная страница

1. module.exports = {
2. block: 'page',
3. mix: { block: 'controller' },
4. title: 'ГПСС',
5. favicon: '/favicon.ico',
6. head: [
7. { elem: 'meta', attrs: { name: 'description', content: '' } },
8. { elem: 'meta', attrs: { name: 'viewport', content: 'width=device-width, initial-scale=1' } },
9. { elem: 'css', url: 'index.min.css' }
10. ],
11. scripts: [{ elem: 'js', url: 'index.min.js' }],
12. mods: { theme: 'islands' },
13. content: [
14. {
15. block: 'row',
16. content: {
17. block: 'header',
18. mix: {
19. block: 'row',
20. elem: 'col',
21. elemMods: {
22. m: **true**
23. }
24. },
25. content: 'Генератор персональных советующих систем'
26. }
27. },
28. {
29. block: 'row',
30. content: {
31. block: 'list',
32. mods: {
33. visible: **true**
34. },
35. js: **true**,
36. content: [
37. {
38. block: 'headline',
39. content: 'Список ваших персональных советующих  
    систем'
40. },
41. {
42. elem: 'content'
43. },
44. {
45. block: 'button',
46. mix: { block: 'list', elem: 'add' },
47. mods: { theme: 'islands', size: 'm', view: 'action' },
48. text: 'Добавить новую'
49. }
50. ]
51. }
52. },
53. {
54. block: 'row',
55. content: {
56. block: 'new-system',
57. mix: {
58. block: 'row',
59. elem: 'col',
60. elemMods: {
61. m: **true**
62. }
63. },
64. js: **true**,
65. content: [
66. {
67. block: 'headline',
68. content: 'Введите имя и цель для новой ПСС'
69. },
70. {
71. block: 'control-group',
72. mix: { block: 'new-system', elem: 'form' },
73. tag: 'form',
74. content: [
75. {
76. block: 'input',
77. mods: { theme: 'islands', size: 'm'},
78. mix: { block: 'new-system', elem: 'name' },
79. required: 'required',
80. placeholder: 'Имя',
81. val: 'Тестовое имя'
82. },
83. {
84. block: 'input',
85. mods: { theme: 'islands', size: 'm' },
86. mix: { block: 'new-system', elem: 'goal' },
87. placeholder: 'Цель',
88. val: 'Тестовая цель'
89. },
90. {
91. block: 'button',
92. mix: { block: 'new-system', elem: 'save' },
93. mods: { theme: 'islands', size: 'm', view: 'action' },
94. attrs: {
95. type: 'submit'
96. },
97. text: 'Сохранить'
98. }
99. ]
100. }
101. ]
102. }
103. },
104. {
105. block: 'row',
106. content: {
107. block: 'settings',
108. tag: 'form',
109. js: **true**,
110. content: [
111. {
112. block: 'headline',
113. content: 'Введите лингвистические переменные'
114. },
115. {
116. block: 'row',
117. content: [
118. {
119. elem: 'col',
120. mods: {
121. m: **true**
122. },
123. content: {
124. block: 'control-group',
125. mix: { block: 'settings', elem: 'ling-var' },
126. content: [
127. {
128. block: 'input',
129. mix: { block: 'settings', elem: 'name' },
130. mods: { theme: 'islands', size: 'm' },
131. required: 'required',
132. placeholder: 'Название'
133. },
134. {
135. block: 'input',
136. mix: { block: 'settings', elem: 'value' },
137. mods: { theme: 'islands', size: 'm' },
138. required: 'required',
139. placeholder: 'Первое значение'
140. },
141. {
142. block: 'input',
143. mods: { theme: 'islands', size: 'm' },
144. mix: { block: 'settings', elem: 'value' },
145. required: 'required',
146. placeholder: 'Второе значение'
147. },
148. {
149. block: 'button',
150. mods: { theme: 'islands', size: 'm' },
151. mix: { block: 'settings', elem: 'add-value' },
152. text: 'Добавить значение'
153. }
154. ]
155. }
156. }
157. ]
158. },
159. {
160. block: 'row',
161. mix: { block: 'settings', elem: 'last-row' },
162. content: [
163. {
164. elem: 'col',
165. mods: {
166. m: **true**
167. },
168. content: [
169. {
170. block: 'button',
171. mods: { theme: 'islands', size: 'm' },
172. mix: { block: 'settings', elem: 'add-variable' },
173. text: 'Добавить переменную'
174. },
175. {
176. block: 'button',
177. mods: { theme: 'islands', size: 'm', view: 'action' },
178. mix: { block: 'settings', elem: 'save' },
179. attrs: {
180. type: 'submit'
181. },
182. text: 'Сохранить'
183. }
184. ]
185. }
186. ]
187. }
188. ]
189. }
190. },
191. {
192. block: 'row',
193. content: {
194. block: 'comparing',
195. js: **true**,
196. content: [
197. {
198. block: 'headline',
199. content: 'Оценка шкал сравнения'
200. },
201. {
202. block: 'row',
203. mix: { block: 'comparing', elem: 'line' },
204. content: {
205. elem: 'col',
206. content: [
207. {
208. block: 'row',
209. content: [
210. {
211. elem: 'col',
212. content: {
213. block: 'comparing',
214. elem: 'tip',
215. content: '«Первое значение»'
216. }
217. },
218. {
219. elem: 'col',
220. content: {
221. block: 'select',
222. mods: { mode: 'radio-check', theme: 'islands', size: 'm' },
223. mix: { block: 'comparing', elem: 'degree' },
224. name: 'select1',
225. text: '—',
226. options: [
227. { val: 1, text: 'более важно, чем' },
228. { val: 2, text: 'менее важно, чем' },
229. { val: 3, text: 'так же важно, как и' }
230. ]
231. }
232. },
233. {
234. elem: 'col',
235. content: {
236. block: 'comparing',
237. elem: 'tip',
238. content: '«Второе значение»'
239. }
240. },
241. {
242. elem: 'col',
243. content: {
244. block: 'select',
245. mods: { mode: 'radio-check', theme: 'islands', size: 'm' },
246. mix: { block: 'comparing', elem: 'degree' },
247. name: 'select2',
248. text: '—',
249. options: [
250. { val: 1, text: 'в 1 раз' },
251. { val: 2, text: 'в 2 раза' },
252. { val: 3, text: 'в 3 раза' },
253. { val: 4, text: 'в 4 раза' },
254. { val: 5, text: 'в 5 раз' },
255. { val: 6, text: 'в 6 раз' },
256. { val: 7, text: 'в 7 раз' },
257. { val: 8, text: 'в 8 раз' },
258. { val: 9, text: 'в 9 раз' }
259. ]
260. }
261. }
262. ]
263. }
264. ]
265. }
266. },
267. {
268. elem: 'save',
269. content: {
270. block: 'button',
271. mods: { theme: 'islands', size: 'm', view: 'action' },
272. text: 'Сохранить'
273. }
274. }
275. ]
276. }
277. },
278. {
279. block: 'row',
280. content: {
281. block: 'data',
282. content: [
283. {
284. block: 'headline',
285. content: 'Загрузка существующих данных'
286. },
287. {
288. block : 'textarea',
289. mods : { theme : 'islands', size : 'm', width : 'available' },
290. placeholder : 'Тут будет шаблон для будущего JSON'
291. },
292. {
293. elem: 'load',
294. content: {
295. block : 'attach',
296. mods : { theme : 'islands', size : 'm' },
297. button : 'Выберите файл с данными',
298. noFileText : 'Файл не выбран'
299. }
300. },
301. {
302. elem: 'save',
303. content: {
304. block: 'button',
305. mods: { theme: 'islands', size: 'm', view: 'action' },
306. text: 'Сохранить'
307. }
308. }
309. ]
310. }
311. }
312. ]
313. };

Б.9. Элемент dom блока i-bem (переопределённые методы)

1. modules.define(
2. 'i-bem\_\_dom',
3. ['i-bem', 'i-bem\_\_internal', 'identify', 'objects',  
   'functions', 'jquery', 'dom'],
4. **function**(provide, BEM, INTERNAL, identify, objects, functions, $, dom) {
6. **var** DOM;
8. $(**function**() {
9. DOM = BEM.decl('i-bem\_\_dom', {
10. findBlocksInside : **function**(elem, block) {
11. **return** **this**.\_findBlocks('find', elem, block);
12. },
14. findBlockInside : **function**(elem, block) {
15. **return** **this**.\_findBlocks('find', elem, block, **true**);
16. },
18. findBlocksOutside : **function**(elem, block) {
19. **return** **this**.\_findBlocks('parents', elem, block);
20. },
22. findBlockOutside : **function**(elem, block) {
23. **return** **this**.\_findBlocks('closest', elem, block)[0] || **null**;
24. },
26. findBlocksOn : **function**(elem, block) {
27. **return** **this**.\_findBlocks('', elem, block);
28. },
30. findBlockOn : **function**(elem, block) {
31. **return** **this**.\_findBlocks('', elem, block, **true**);
32. },
34. \_findBlocks : **function**(select, elem, block, onlyFirst) {
35. **if**(!block) {
36. block = elem;
37. elem = undef;
38. }
40. **var** ctxElem = elem?
41. (**typeof** elem === 'string'? **this**.findElem(elem) : elem) :
42. **this**.domElem,
43. isSimpleBlock = **typeof** block === 'string',
44. blockName = isSimpleBlock? block : (block.block  
    || block.blockName),
45. selector = '.' +
46. (isSimpleBlock?
47. buildClass(blockName) :
48. buildClass(blockName, block.modName, block.modVal)) +
49. (onlyFirst? ':first' : ''),
50. domElems = ctxElem.filter(selector);
52. select && (domElems = domElems.add(ctxElem[select](selector)));
54. **if**(onlyFirst) {
55. **return** domElems[0]? initBlock(blockName, domElems.eq(0),  
    undef, **true**).\_init() : **null**;
56. }
58. **var** res = [],
59. uniqIds = {};
61. domElems.each(**function**(i, domElem) {
62. **var** block = initBlock(blockName, $(domElem),  
    undef, **true**).\_init();
63. **if**(!uniqIds[block.\_uniqId]) {
64. uniqIds[block.\_uniqId] = **true**;
65. res.push(block);
66. }
67. });
69. **return** res;
70. },
72. bindToDomElem : **function**(domElem, event, data, fn) {
73. **if**(functions.isFunction(data)) {
74. fn = data;
75. data = undef;
76. }
78. fn?
79. domElem.bind(
80. **this**.\_buildEventName(event),
81. data,
82. $.proxy(fn, **this**)) :
83. objects.each(event, **function**(fn, event) {
84. **this**.bindToDomElem(domElem, event, data, fn);
85. }, **this**);
87. **return** **this**;
88. },
90. bindTo : **function**(elem, event, data, fn) {
91. **var** len = arguments.length;
92. **if**(len === 3) {
93. **if**(functions.isFunction(data)) {
94. fn = data;
95. **if**(**typeof** event === 'object') {
96. data = event;
97. event = elem;
98. elem = **this**.domElem;
99. }
100. }
101. } **else** **if**(len === 2) {
102. **if**(functions.isFunction(event)) {
103. fn = event;
104. event = elem;
105. elem = **this**.domElem;
106. } **else** **if**(!(**typeof** elem === 'string' || elem **instanceof** $)) {
107. data = event;
108. event = elem;
109. elem = **this**.domElem;
110. }
111. } **else** **if**(len === 1) {
112. event = elem;
113. elem = **this**.domElem;
114. }
116. **typeof** elem === 'string' && (elem = **this**.elem(elem));
118. **return** **this**.bindToDomElem(elem, event, data, fn);
119. },
121. unbindFromDomElem : **function**(domElem, event, fn) {
122. **if**(**typeof** event === 'string') {
123. event = **this**.\_buildEventName(event);
124. fn?
125. domElem.unbind(event, fn) :
126. domElem.unbind(event);
127. } **else** {
128. objects.each(event, **function**(fn, event) {
129. **this**.unbindFromDomElem(domElem, event, fn);
130. }, **this**);
131. }
133. **return** **this**;
134. },
136. unbindFrom : **function**(elem, event, fn) {
137. **var** argLen = arguments.length;
138. **if**(argLen === 1) {
139. event = elem;
140. elem = **this**.domElem;
141. } **else** **if**(argLen === 2 && functions.isFunction(event)) {
142. fn = event;
143. event = elem;
144. elem = **this**.domElem;
145. } **else** **if**(**typeof** elem === 'string') {
146. elem = **this**.elem(elem);
147. }
149. **return** **this**.unbindFromDomElem(elem, event, fn);
150. },
152. \_buildEventName : **function**(event) {
153. **return** event.indexOf(' ') > 1?
154. event.split(' ').map(**function**(e) {
155. **return** **this**.\_buildOneEventName(e);
156. }, **this**).join(' ') :
157. **this**.\_buildOneEventName(event);
158. },
160. \_buildOneEventName : **function**(event) {
161. **var** eventNameCache = **this**.\_eventNameCache;
163. **if**(event **in** eventNameCache) **return** eventNameCache[event];
165. **var** uniq = '.' + **this**.\_uniqId;
167. **if**(event.indexOf('.') < 0) **return** eventNameCache[event] = event + uniq;
169. **var** lego = '.bem\_' + **this**.\_\_self.\_name;
171. **return** eventNameCache[event] =  
     event.split('.').map(**function**(e, i) {
172. **return** i === 0? e + lego : lego + '\_' + e;
173. }).join('') + uniq;
174. },
176. \_ctxEmit : **function**(e, data) {
177. **this**.\_\_base.apply(**this**, arguments);
179. **var** \_this = **this**,
180. storage =  
     liveEventCtxStorage[\_this.\_\_self.\_buildCtxEventName(e.type)],
181. ctxIds = {};
183. storage && \_this.domElem.each(**function**(\_, ctx) {
184. **var** counter = storage.counter;
185. **while**(ctx && counter) {
186. **var** ctxId = identify(ctx, **true**);
187. **if**(ctxId) {
188. **if**(ctxIds[ctxId]) **break**;
189. **var** storageCtx = storage.ctxs[ctxId];
190. **if**(storageCtx) {
191. objects.each(storageCtx, **function**(handler) {
192. handler.fn.call(
193. handler.ctx || \_this,
194. e,
195. data);
196. });
197. counter--;
198. }
199. ctxIds[ctxId] = **true**;
200. }
201. ctx = ctx.parentNode || domNodesToParents[ctxId];
202. }
203. });
204. },
206. setMod : **function**(elem, modName, modVal) {
207. **if**(elem && **typeof** modVal !== 'undefined' && elem.length > 1) {
208. **var** \_this = **this**;
209. elem.each(**function**() {
210. **var** item = $(**this**);
211. item.\_\_bemElemName = elem.\_\_bemElemName;
212. \_this.setMod(item, modName, modVal);
213. });
214. **return** \_this;
215. }
216. **return** **this**.\_\_base(elem, modName, modVal);
217. },
219. \_extractModVal : **function**(modName, elem, elemName) {
220. **var** domNode = (elem || **this**.domElem)[0],
221. matches;
223. domNode &&
224. (matches = domNode.className
225. .match(**this**.\_\_self.\_buildModValRE(modName, elemName || elem)));
227. **return** matches? matches[2] || **true** : '';
228. },
230. \_extractMods : **function**(modNames, elem) {
231. **var** res = {},
232. extractAll = !modNames.length,
233. countMatched = 0;
235. ((elem || **this**.domElem)[0].className
236. .match(**this**.\_\_self.\_buildModValRE(
237. '(' + (extractAll? NAME\_PATTERN : modNames.join('|')) + ')',
238. elem,
239. 'g')) || []).forEach(**function**(className) {
240. **var** matches = className.match(EXTRACT\_MODS\_RE);
241. res[matches[1]] = matches[2] || **true**;
242. ++countMatched;
243. });
245. // empty modifier values are not reflected in classes; they must be filled with empty values
246. countMatched < modNames.length && modNames.forEach(**function**(  
     modName) {
247. modName **in** res || (res[modName] = '');
248. });
250. **return** res;
251. },
253. \_onSetMod : **function**(modName, modVal, oldModVal, elem, elemName) {
254. **if**(modName !== 'js' || modVal !== '') {
255. **var** \_self = **this**.\_\_self,
256. classPrefix = \_self.\_buildModClassPrefix(modName,  
     elemName),
257. classRE = \_self.\_buildModValRE(modName, elemName),
258. needDel = modVal === '' || modVal === **false**;
260. (elem || **this**.domElem).each(**function**() {
261. **var** className = **this**.className,
262. modClassName = classPrefix;
264. modVal !== **true** && (modClassName += MOD\_DELIM + modVal);
266. (oldModVal === **true**?
267. classRE.test(className) :
268. (' ' + className).indexOf(' ' + classPrefix +  
     MOD\_DELIM) > -1)?
269. **this**.className = className.replace(
270. classRE,
271. (needDel? '' : '$1' + modClassName)) :
272. needDel || $(**this**).addClass(modClassName);
273. });
275. elemName && **this**
276. .dropElemCache(elemName, modName, oldModVal)
277. .dropElemCache(elemName, modName, modVal);
278. }
280. **this**.\_\_base.apply(**this**, arguments);
281. },
283. findElem : **function**(ctx, names, modName, modVal, strictMode) {
284. **if**(**typeof** ctx === 'string') {
285. strictMode = modVal;
286. modVal = modName;
287. modName = names;
288. names = ctx;
289. ctx = **this**.domElem;
290. }
292. **if**(**typeof** modName === 'boolean') {
293. strictMode = modName;
294. modName = undef;
295. }
297. names = names.split(' ');
299. **var** \_self = **this**.\_\_self,
300. modPostfix = buildModPostfix(modName, modVal),
301. selectors = [],
302. keys = names.map(**function**(name) {
303. selectors.push(\_self.buildSelector(name, modName, modVal));
304. **return** name + modPostfix;
305. }),
306. isSingleName = keys.length === 1,
307. res = findDomElem(ctx, selectors.join(','));
309. // caching results if possible
310. ctx === **this**.domElem &&
311. selectors.forEach(**function**(selector, i) {
312. (**this**.\_elemCache[keys[i]] = isSingleName? res :  
     res.filter(selector))
313. .\_\_bemElemName = names[i];
314. }, **this**);
316. **return** strictMode? **this**.\_filterFindElemResults(res) : res;
317. },
319. \_filterFindElemResults : **function**(res) {
320. **var** blockSelector = **this**.buildSelector(),
321. domElem = **this**.domElem;
322. **return** res.filter(**function**() {
323. **return** domElem.index($(**this**).closest(blockSelector)) > -1;
324. });
325. },
327. \_elem : **function**(name, modName, modVal) {
328. **return** **this**.\_elemCache[name + buildModPostfix(modName, modVal)] ||
329. **this**.findElem(name, modName, modVal);
330. },
332. elem : **function**(names, modName, modVal) {
333. **if**(arguments.length === 2) {
334. modVal = **true**;
335. }
337. **if**(modName && **typeof** modName !== 'string') {
338. modName.\_\_bemElemName = names;
339. **return** modName;
340. }
342. **if**(names.indexOf(' ') < 0) {
343. **return** **this**.\_elem(names, modName, modVal);
344. }
346. **var** res = $([]);
347. names.split(' ').forEach(**function**(name) {
348. res = res.add(**this**.\_elem(name, modName, modVal));
349. }, **this**);
350. **return** res;
351. },
353. closestElem : **function**(ctx, elemName) {
354. **return** ctx.closest(**this**.buildSelector(elemName));
355. },
357. dropElemCache : **function**(names, modName, modVal) {
358. **if**(names) {
359. **var** modPostfix = buildModPostfix(modName, modVal);
360. names.indexOf(' ') < 0?
361. **delete** **this**.\_elemCache[names + modPostfix] :
362. names.split(' ').forEach(**function**(name) {
363. **delete** **this**.\_elemCache[name + modPostfix];
364. }, **this**);
365. } **else** {
366. **this**.\_elemCache = {};
367. }
369. **return** **this**;
370. },
372. elemParams : **function**(elem) {
373. **var** elemName;
374. **if**(**typeof** elem === 'string') {
375. elemName = elem;
376. elem = **this**.elem(elem);
377. } **else** {
378. elemName = **this**.\_\_self.\_extractElemNameFrom(elem);
379. }
381. **return** extractParams(elem[0])[**this**.\_\_self.buildClass(elemName)] || {};
382. },
384. elemify : **function**(elem, elemName) {
385. (elem = $(elem)).\_\_bemElemName = elemName;
386. **return** elem;
387. },
389. containsDomElem : **function**(ctx, domElem) {
390. **if**(arguments.length === 1) {
391. domElem = ctx;
392. ctx = **this**.domElem;
393. }
395. **return** dom.contains(ctx, domElem);
396. },
398. buildSelector : **function**(elem, modName, modVal) {
399. **return** **this**.\_\_self.buildSelector(elem, modName, modVal);
400. },
402. \_destruct : **function**() {
403. **var** \_this = **this**,
404. \_self = \_this.\_\_self;
406. \_this.\_needSpecialUnbind &&   
     self.doc.add(\_self.win).unbind('.' + \_this.\_uniqId);
408. \_this.\_\_base();
410. **delete** uniqIdToBlock[\_this.un().\_uniqId];
411. }
413. }, {
414. scope : $('body'),
416. init : **function**(ctx) {
417. **if**(**typeof** ctx === 'string') {
418. ctx = $(ctx);
419. } **else** **if**(!ctx) ctx = DOM.scope;
421. **var** uniqInitId = identify();
422. findDomElem(ctx, BEM\_SELECTOR).each(**function**() {
423. initBlocks($(**this**), uniqInitId);
424. });
426. **this**.\_runInitFns();
428. **return** ctx;
429. },
431. destruct : **function**(ctx, excludeSelf) {
432. **this**.\_destruct(ctx, excludeSelf, **true**);
433. },
435. detach : **function**(ctx, excludeSelf) {
436. **this**.\_destruct(ctx, excludeSelf);
437. },
439. update : **function**(ctx, content) {
440. **this**.destruct(ctx, **true**);
441. **return** **this**.init(ctx.html(content));
442. },
444. replace : **function**(ctx, content) {
445. **var** prev = ctx.prev(),
446. parent = ctx.parent();
448. content = getJqueryCollection(content);
450. **this**.destruct(ctx);
452. **return** **this**.init(prev.length?
453. content.insertAfter(prev) :
454. content.prependTo(parent));
455. },
457. append : **function**(ctx, content) {
458. **return** **this**.init(getJqueryCollection(content).appendTo(ctx));
459. },
461. prepend : **function**(ctx, content) {
462. **return** **this**.init(getJqueryCollection(content).prependTo(ctx));
463. },
465. before : **function**(ctx, content) {
466. **return** **this**.init(getJqueryCollection(content).insertBefore(ctx));
467. },
469. after : **function**(ctx, content) {
470. **return** **this**.init(getJqueryCollection(content).insertAfter(ctx));
471. },
473. liveBindTo : **function**(to, event, callback, invokeOnInit) {
474. **if**(!event || functions.isFunction(event)) {
475. callback = event;
476. event = to;
477. to = undef;
478. }
480. **if**(!to || **typeof** to === 'string') {
481. to = { elem : to };
482. }
484. **if**(to.elem && to.elem.indexOf(' ') > 0) {
485. to.elem.split(' ').forEach(**function**(elem) {
486. **this**.\_liveClassBind(
487. **this**.buildClass(elem, to.modName, to.modVal),
488. event,
489. callback,
490. invokeOnInit);
491. }, **this**);
492. **return** **this**;
493. }
495. **return** **this**.\_liveClassBind(
496. **this**.buildClass(to.elem, to.modName, to.modVal),
497. event,
498. callback,
499. invokeOnInit);
500. },
502. liveUnbindFrom : **function**(elem, event, callback) {
504. **if**(!event || functions.isFunction(event)) {
505. callback = event;
506. event = elem;
507. elem = undef;
508. }
510. **if**(elem && elem.indexOf(' ') > 1) {
511. elem.split(' ').forEach(**function**(elem) {
512. **this**.\_liveClassUnbind(
513. **this**.buildClass(elem),
514. event,
515. callback);
516. }, **this**);
517. **return** **this**;
518. }
520. **return** **this**.\_liveClassUnbind(
521. **this**.buildClass(elem),
522. event,
523. callback);
524. },
526. buildSelector : **function**(elem, modName, modVal) {
527. **return** '.' + **this**.buildClass(elem, modName, modVal);
528. }
529. });
531. $.fn.bem = **function**(blockName, params) {
532. **return** initBlock(blockName, **this**, params, **true**).\_init();
533. };
535. provide(DOM);
537. });
539. });

Б.10. Файл конфигураций ENB

1. **var** techs = {
2. // essential
3. fileProvider: require('enb/techs/file-provider'),
4. fileMerge: require('enb/techs/file-merge'),
6. // optimization
7. borschik: require('enb-borschik/techs/borschik'),
9. // css
10. stylus: require('enb-stylus/techs/stylus'),
12. // js
13. browserJs: require('enb-js/techs/browser-js'),
15. // bemhtml
16. bemhtml: require('enb-bemxjst/techs/bemhtml'),
17. bemjsonToHtml: require('enb-bemxjst/techs/bemjson-to-html')
18. },
19. enbBemTechs = require('enb-bem-techs'),
20. levels = [
21. { path: 'libs/bem-core/common.blocks', check: **false** },
22. { path: 'libs/bem-core/desktop.blocks', check: **false** },
23. { path: 'libs/bem-components/common.blocks', check: **false** },
24. { path: 'libs/bem-components/desktop.blocks', check: **false** },
25. { path: 'libs/bem-components/design/common.blocks', check: **false** },
26. { path: 'libs/bem-components/design/desktop.blocks', check: **false** },
27. 'common.blocks',
28. 'desktop.blocks'
29. ];
31. module.exports = **function**(config) {
32. **var** isProd = process.env.YENV === 'production';
34. config.nodes('\*.bundles/\*', **function**(nodeConfig) {
35. nodeConfig.addTechs([
36. // essential
37. [enbBemTechs.levels, { levels: levels }],
38. [techs.fileProvider, { target: '?.bemjson.js' }],
39. [enbBemTechs.bemjsonToBemdecl],
40. [enbBemTechs.deps],
41. [enbBemTechs.files],
43. // css
44. [techs.stylus, {
45. target: '?.no-grid.css',
46. sourcemap: **false**,
47. autoprefixer: {
48. browsers: ['ie >= 10', 'last 2 versions', 'opera 12.1', '> 2%']
49. }
50. }],
52. [require('bem-grid').enb, {
53. config : {
54. maxWidth : '800px',
55. gutter : '10px',
56. flex : 'flex'
57. },
58. source: '?.no-grid.css'
59. }],
61. // bemhtml
62. [techs.bemhtml, { sourceSuffixes: ['bemhtml', 'bemhtml.js'] }],
64. // html
65. [techs.bemjsonToHtml],
67. // client bemhtml
68. [enbBemTechs.depsByTechToBemdecl, {
69. target: '?.bemhtml.bemdecl.js',
70. sourceTech: 'js',
71. destTech: 'bemhtml'
72. }],
73. [enbBemTechs.deps, {
74. target: '?.bemhtml.deps.js',
75. bemdeclFile: '?.bemhtml.bemdecl.js'
76. }],
77. [enbBemTechs.files, {
78. depsFile: '?.bemhtml.deps.js',
79. filesTarget: '?.bemhtml.files',
80. dirsTarget: '?.bemhtml.dirs'
81. }],
82. [techs.bemhtml, {
83. target: '?.browser.bemhtml.js',
84. filesTarget: '?.bemhtml.files',
85. sourceSuffixes: ['bemhtml', 'bemhtml.js'],
86. forceBaseTemplates: **true**
87. }],
89. // js
90. [techs.browserJs, { includeYM: **true** }],
91. [techs.fileMerge, {
92. target: '?.js',
93. sources: ['?.browser.js', '?.browser.bemhtml.js']
94. }],
96. // borschik
97. [techs.borschik, { source: '?.js', target: '?.min.js', minify:  
    isProd }],
98. [techs.borschik, { source: '?.css', target: '?.min.css',  
    minify: isProd }]
99. ]);
101. nodeConfig.addTargets([/\* '?.bemtree.js', \*/ '?.html', '?.min.css', '?.min.js']);
102. });
103. };