תוכנית יישום 24 שעות

סקירת זמנים וחלוקת משימות

```
┌─────────────────────────────────────────────┐
│ שעות 0-4: תשתית ו Modelים (4 שעות)          │
├─────────────────────────────────────────────┤
│ • הרחבת סכמות קיימות (1 שעה)                │
│ • יצירת מודלים חדשים (1 שעה)                │
│ • API endpoints בסיסיים (2 שעות)            │
└─────────────────────────────────────────────┘
↓
┌─────────────────────────────────────────────┐
│ שעות 4-12: ממשק משתמש (8 שעות)              │
├─────────────────────────────────────────────┤
│ • דף תכנון שבועי (3 שעות)                   │
│ • דף הזנת טיפולים (2 שעות)                  │
│ • לוח בקרה משפחתי (2 שעות)                  │
│ • מעקב מצב רוח (1 שעה)                      │
└─────────────────────────────────────────────┘
↓
┌─────────────────────────────────────────────┐
│ שעות 12-18: לוגיקה עסקית (6 שעות)           │
├─────────────────────────────────────────────┤
│ • מנוע תכנון בסיסי (3 שעות)                 │
│ • חלוקת משימות משפחתיות (2 שעות)            │
│ • אוטומציות פשוטות (1 שעה)                  │
└─────────────────────────────────────────────┘
↓
┌─────────────────────────────────────────────┐
│ שעות 18-22: אינטגרציה ובדיקות (4 שעות)      │
├─────────────────────────────────────────────┤
│ • חיבור Monday.com (2 שעות)                 │
│ • בדיקות ותיקונים (2 שעות)                  │
└─────────────────────────────────────────────┘
↓
┌─────────────────────────────────────────────┐
```

│ השלמות ו-Deploy (שעות 22-24: 2 שעות) │
├──────────────────────────────────────┤
│ • תיעוד מהיר (30 דקות) │
│ • הגדרות production (30 דקות) │
│ • בדיקה סופית (1 שעה) │
└──────────────────────────────────────┘

שעות 0-4: תשתית נתונים

1. הרחבת סכמות (30 דקות)

```javascript
// updateSchemas.js
const updateTaskSchema = async () => {
  // הוסף שדות חדשים ל-Task
  await db.collection('tasks').updateMany(
    {},
    {
      $set: {
        estimatedDuration: 30,
        cognitiveLoad: 'medium',
        locationFlexibility: 'office_only',
        energyLevel: 'medium',
        bufferDays: 2,
        suitableFor: ['parent'],
        parentApprovalRequired: true
      }
    }
  );
};

// models/Task.js - הוסף לסכמה הקיימת
TaskSchema.add({
  estimatedDuration: { type: Number, default: 30 },
  cognitiveLoad: { type: String, enum: ['low', 'medium', 'high'], default: 'medium' },
  locationFlexibility: { type: String, enum: ['office_only', 'remote_possible', 'anywhere'], default: 'office_only' },
  energyLevel: { type: String, enum: ['low', 'medium', 'high'], default: 'medium' },
  bufferDays: { type: Number, default: 2 },
  suitableFor: [{ type: String, enum: ['parent', 'teen16', 'teen14'] }],
  parentApprovalRequired: { type: Boolean, default: true }
});
```

2. יצירת מודלים חדשים (30 דקות)

javascript

```javascript
// models/WeeklySchedule.js
const WeeklyScheduleSchema = new Schema({
  userId: String,
  weekStartDate: Date,
  fixedBlocks: [{
    day: Number,
    startTime: String,
    endTime: String,
    type: String,
    isFixed: Boolean
  }],
  scheduledTasks: [{
    taskId: String,
    day: Number,
    startTime: String,
    endTime: String,
    status: String
  }]
});

// models/FamilyMember.js
const FamilyMemberSchema = new Schema({
  householdId: String,
  name: String,
  age: Number,
  role: String,
  capabilities: [String],
  maxTasksPerDay: Number
});

// models/DailyMood.js
const DailyMoodSchema = new Schema({
  userId: String,
  date: Date,
  mood: Number,
  energy: Number,
  stress: Number,
  notes: String
});
```

3. API Endpoints בסיסיים (2 שעות)

javascript

```javascript
// api/scheduling.js
router.post('/schedule/save-treatments', async (req, res) => {
  const { userId, treatments } = req.body;

  const schedule = await WeeklySchedule.findOneAndUpdate(
    { userId, weekStartDate: startOfWeek(new Date()) },
    {
      fixedBlocks: treatments.map(t => ({
        ...t,
        type: 'treatment',
        isFixed: true
      }))
    },
    { upsert: true, new: true }
  );

  res.json({ success: true, schedule });
});

router.post('/schedule/generate-basic', async (req, res) => {
  const { userId } = req.body;

  // אלגוריתם פשוט - First Fit
  const tasks = await Task.find({
    userId,
    status: { $ne: 'completed' },
    deadline: { $lte: addDays(new Date(), 14) }
  }).sort({ deadline: 1 });

  const schedule = await generateBasicSchedule(userId, tasks);
  res.json({ success: true, schedule });
});

router.post('/family/assign-task', async (req, res) => {
  const { taskId, memberId } = req.body;

  const task = await Task.findByIdAndUpdate(
    taskId,
    { assignedFamilyMember: memberId },
    { new: true }
  );

  res.json({ success: true, task });
});

router.post('/mood/check-in', async (req, res) => {
```

```
  const { userId, mood, energy, stress, notes } = req.body;

  const checkIn = await DailyMood.create({
    userId,
    date: new Date(),
    mood,
    energy,
    stress,
    notes
  });

  res.json({ success: true, checkIn });
});
```

שעות 4-12: ממשק משתמש

סקריפט התקנה מהירה

bash

```
#!/bin/bash
# quick-ui-setup.sh

# התקן תלויות UI
npm install react-beautiful-dnd @dnd-kit/sortable date-fns

# צור מבנה תיקיות
mkdir -p src/components/scheduling
mkdir -p src/components/family
mkdir -p src/components/mood

# העתק תבניות בסיסיות
echo "Creating UI templates..."

# תבנית WeeklyPlanner
cat > src/components/scheduling/WeeklyPlanner.jsx << 'EOF'
import React, { useState, useEffect } from 'react';
import { DragDropContext, Droppable, Draggable } from 'react-beautiful-dnd';

function WeeklyPlanner() {
  const [schedule, setSchedule] = useState(null);
  const [treatments, setTreatments] = useState([]);
  const days = ['ראשון', 'שני', 'שלישי', 'רביעי', 'חמישי'];
  const hours = Array.from({ length: 11 }, (_, i) => `${i + 8}:00`);

  const handleDragEnd = (result) => {
    if (!result.destination) return;

    // לוגיקת גרירה
    console.log('Task moved:', result);
  };

  return (
    <DragDropContext onDragEnd={handleDragEnd}>
      <div className="weekly-planner p-4">
        <h2 className="text-2xl font-bold mb-4">תכנון שבועי</h2>

        <div className="grid grid-cols-6 gap-2">
          {/* עמודת שעות */}
          <div>
            <div className="h-12"></div>
            {hours.map(hour => (
              <div key={hour} className="h-20 flex items-center justify-end pr-2">
                {hour}
              </div>
            ))}
```

```
          </div>

          {/* עמודות ימים */}
          {days.map((day, dayIndex) => (
            <div key={day}>
              <div className="h-12 font-semibold text-center">
                {day}
                {dayIndex === 4 && <span className="text-red-500 text-sm"> (טיפולים)</span>}
              </div>

              <Droppable droppableId={`day-${dayIndex}`}>
                {(provided) => (
                  <div
                    ref={provided.innerRef}
                    {...provided.droppableProps}
                    className="space-y-1"
                  >
                    {/* כאן יופיעו המשימות */}
                    {provided.placeholder}
                  </div>
                )}
              </Droppable>
            </div>
          ))}
        </div>

        <div className="mt-6 flex gap-4">
          <button className="bg-blue-500 text-white px-4 py-2 rounded">
            תכנון אוטומטי
          </button>
          <button className="bg-green-500 text-white px-4 py-2 rounded">
            ייצוא ללוח שנה
          </button>
        </div>
      </div>
    </DragDropContext>
  );
}# CalmPlan - מדריך פיתוח מקיף למערכת תכנון אדפטיבית חכמה

## תוכן עניינים
1. [סקירה כללית](#סקירה-כללית)
2. [ארכיטקטורת המערכת](#ארכיטקטורת-המערכת)
3. [מודלי נתונים מורחבים](#מודלי-נתונים-מורחבים)
4. [ממשק משתמש - רכיבים מרכזיים](#ממשק-משתמש---רכיבים-מרכזיים)
5. [לוגיקה עסקית ואלגוריתמים](#לוגיקה-עסקית-ואלגוריתמים)
6. [אינטגרציות](#אינטגרציות)
7. [תוכנית יישום 24 שעות](#תוכנית-יישום-24-שעות)
```

8. [חלוקת עבודה בין כלים](#חלוקת-עבודה-בין-כלים)
9. [נספחים וקוד לדוגמה](#נספחים-וקוד-לדוגמה)

סקירה כללית

החזון
יצירת מערכת ניהול מרכזית וחכמה (CRM+) המאחדת את כלל הפעילות העסקית והאישית, עם דגש מיוחד על:
- ניהול זמן אדפטיבי המתחשב במגבלות אישיות (ADHD, מצבי חירום משפחתיים)
- תכנון חכם המתאים משימות לרמות אנרגיה וזמינות
- שילוב משפחתי עם חלוקת עומס הוגנת
- מעקב אחר רווחה נפשית ומניעת שחיקה

אתגרים ייחודיים
- ניהול משרד הנהלת חשבונות עצמאי עם דדליינים קשיחים
- התמודדות עם מצב חירום משפחתי (בן זוג בשיקום)
- ניהול בית גדול (200 מ"ר) וגינה (חצי דונם)
- שילוב 2 ילדים (14, 16) בעזרה מבלי להעמיס
- עבודה עם ADHD הדורשת מערכת פשוטה ואינטואיטיבית

ארכיטקטורת המערכת

שכבות המערכת

```
┌─────────────────────────────────────────┐
│         ממשק משתמש (React)              │
│  • תצוגת שבוע דינמית                    │
│  • לוח משימות משפחתי                    │
│  • מעקב מצב רוח                         │
└─────────────────────────────────────────┘
                    │
┌─────────────────────────────────────────┐
│        שכבת לוגיקה (Node.js)            │
│  • מנוע תכנון חכם                       │
│  • מנוע למידה והתאמה                    │
│  • מאזן עומס משפחתי                     │
└─────────────────────────────────────────┘
                    │
┌─────────────────────────────────────────┐
│        אינטגרציות חיצוניות              │
│  • Monday.com (דו-כיווני)               │
```

```
│  • Google Calendar                    │
│  • PriceWise                      │
└──────────────────────────────────────────────────────────────┘
│
┌──────────────────────────────────────────────────────────────┐
│         מסד נתונים (MongoDB)          │
│  • לקוחות, משימות, לוחות זמנים          │
│  • פרופילי משפחה, היסטוריית ביצועים      │
└──────────────────────────────────────────────────────────────┘
```

מודלי נתונים מורחבים

1. הרחבת מודל Task
```javascript
// שדות חדשים למודל Task הקיים
const TaskSchemaExtensions = {
  // תכנון וזמנים
  estimatedDuration: {
    type: Number,
    default: 30,
    description: "משך ביצוע משוער בדקות"
  },
  actualDuration: {
    type: Number,
    description: "משך ביצוע בפועל - למטרות למידה"
  },

  // מאפייני ביצוע
  cognitiveLoad: {
    type: String,
    enum: ['low', 'medium', 'high'],
    default: 'medium',
    description: "רמת המאמץ הקוגניטיבי הנדרש"
  },
  energyLevel: {
    type: String,
    enum: ['low', 'medium', 'high'],
    default: 'medium',
    description: "רמת האנרגיה הנדרשת"
  },

  // גמישות ואילוצים
  locationFlexibility: {
    type: String,
    enum: ['office_only', 'remote_possible', 'mobile_friendly', 'anywhere'],
    default: 'office_only'
  },
  splittable: {
    type: Boolean,
    default: false,
    description: "האם ניתן לפצל את המשימה"
  },
```

```
  // תכנון ותלויות
  bufferDays: {
    type: Number,
    default: 2,
    description: "כמה ימים לפני הדדליין להתחיל"
  },
  dependencies: [{
    type: Schema.Types.ObjectId,
    ref: 'Task'
  }],
  preferredTimeOfDay: {
    type: String,
    enum: ['early_morning', 'morning', 'afternoon', 'evening', 'night', 'anytime'],
    default: 'anytime'
  },

  // משפחה ואצילה
  suitableFor: [{
    type: String,
    enum: ['parent', 'teen16', 'teen14', 'anyone']
  }],
  parentApprovalRequired: {
    type: Boolean,
    default: true
  },
  assignedFamilyMember: {
    type: Schema.Types.ObjectId,
    ref: 'FamilyMember'
  }
}
```

3. מאזן עומס משפחתי

```javascript
// FamilyLoadBalancer.js
class FamilyLoadBalancer {
  constructor(familyMembers) {
    this.familyMembers = familyMembers;
    this.currentLoad = {};
    this.capabilities = {};
    this.preferences = {};

    this.initializeLoadTracking();
  }

  initializeLoadTracking() {
    this.familyMembers.forEach(member => {
      this.currentLoad[member.id] = {
```

```
      daily: {},
      weekly: 0,
      tasks: []
    };
    this.capabilities[member.id] = member.capabilities;
    this.preferences[member.id] = member.preferences || {};
  });
}

async suggestAssignment(task, timeSlot) {
  const eligibleMembers = this.filterEligibleMembers(task);
  const scoredMembers = [];

  for (const member of eligibleMembers) {
    const score = await this.calculateAssignmentScore(member, task, timeSlot);
    scoredMembers.push({ member, score, breakdown: score.breakdown });
  }

  // מיון לפי ציון מהגבוה לנמוך
  scoredMembers.sort((a, b) => b.score.total - a.score.total);

  // החזר את 3 ההתאמות הטובות ביותר
  return scoredMembers.slice(0, 3).map(item => ({
    member: item.member,
    score: item.score.total,
    reasons: this.explainAssignment(item),
    warnings: this.checkAssignmentWarnings(item.member, task)
  }));
}

filterEligibleMembers(task) {
  return this.familyMembers.filter(member => {
    // בדיקת גיל מתאים
    if (!task.suitableFor.includes(member.role)) {
      return false;
    }

    // בדיקת יכולות נדרשות
    if (task.requiredCapabilities) {
      const hasAllCapabilities = task.requiredCapabilities.every(
        cap => member.capabilities.includes(cap)
      );
      if (!hasAllCapabilities) return false;
    }

    // בדיקת זמינות בסיסית
    if (member.currentLoad.daily[task.date] >= member.maxTasksPerDay) {
```

```
      return false;
    }

    return true;
  });
}

async calculateAssignmentScore(member, task, timeSlot) {
  const score = {
    availability: 0,
    capability: 0,
    workload: 0,
    preference: 0,
    history: 0,
    fairness: 0,
    breakdown: {}
  };

  // זמינות בזמן הספציפי
  if (this.isMemberAvailable(member, timeSlot)) {
    score.availability = 30;
    score.breakdown.availability = 'זמין בזמן המבוקש';
  } else {
    score.availability = 0;
    score.breakdown.availability = 'לא זמין בזמן המבוקש';
  }

  // התאמת יכולות
  const capabilityMatch = this.calculateCapabilityMatch(member, task);
  score.capability = capabilityMatch * 25;
  score.breakdown.capability = `${capabilityMatch * 100}% התאמת יכולות`;

  // עומס נוכחי
  const currentLoadRatio = this.getCurrentLoadRatio(member);
  score.workload = (1 - currentLoadRatio) * 20;
  score.breakdown.workload = `עומס נוכחי: ${Math.round(currentLoadRatio * 100)}%`;

  // העדפות אישיות
  if (member.preferences?.favoriteTaskTypes?.includes(task.type)) {
    score.preference = 15;
    score.breakdown.preference = 'סוג משימה מועדף';
  }

  // היסטוריית ביצועים
  const historicalSuccess = await this.getHistoricalSuccess(member, task.type);
  score.history = historicalSuccess * 10;
  score.breakdown.history = `הצלחה היסטורית: ${Math.round(historicalSuccess * 100)}%`;
```

```
  // הוגנות - האם החבר הזה כבר לקח הרבה משימות השבוע?
  const fairnessScore = this.calculateFairness(member);
  score.fairness = fairnessScore * 10;
  score.breakdown.fairness = `ציון הוגנות: ${Math.round(fairnessScore * 100)}%`;

  // חישוב ציון כולל
  score.total = Object.entries(score)
    .filter(([key]) => key !== 'total' && key !== 'breakdown')
    .reduce((sum, [_, value]) => sum + value, 0);

  return score;
}

checkAssignmentWarnings(member, task) {
  const warnings = [];

  // בדיקת עומס יתר פוטנציאלי
  const projectedLoad = this.getProjectedLoad(member, task);
  if (projectedLoad > 0.8) {
    warnings.push({
      type: 'high_load',
      message: `${member.name} יהיה עמוס מאוד אם ייקח משימה זו`,
      severity: 'medium'
    });
  }

  // בדיקת משימות בימי בית ספר
  if (member.schoolDays?.includes(task.dayOfWeek) && task.estimatedDuration > 60) {
    warnings.push({
      type: 'school_conflict',
      message: 'משימה ארוכה ביום לימודים',
      severity: 'low'
    });
  }

  // בדיקת מורכבות מול גיל
  if (member.age < 16 && task.complexity === 'complex') {
    warnings.push({
      type: 'complexity',
      message: 'המשימה עשויה להיות מורכבת מדי',
      severity: 'high'
    });
  }

  return warnings;
}
```

```
async distributeWeeklyTasks(unassignedTasks) {
  const distribution = {
    assignments: [],
    unassigned: [],
    statistics: {}
  };

  // מיון משימות לפי דחיפות וחשיבות
  const sortedTasks = this.prioritizeFamilyTasks(unassignedTasks);

  for (const task of sortedTasks) {
    const suggestions = await this.suggestAssignment(task);

    if (suggestions.length > 0 && suggestions[0].score > 50) {
      // הקצה למועמד הטוב ביותר
      const assignment = {
        task: task,
        assignedTo: suggestions[0].member,
        score: suggestions[0].score,
        reasons: suggestions[0].reasons,
        requiresApproval: task.parentApprovalRequired
      };

      distribution.assignments.push(assignment);
      this.updateMemberLoad(suggestions[0].member, task);
    } else {
      // לא נמצאה התאמה טובה
      distribution.unassigned.push({
        task: task,
        reason: suggestions.length === 0
          ? 'אין חברי משפחה מתאימים'
          : 'ציון התאמה נמוך מדי',
        suggestions: suggestions
      });
    }
  }

  // חישוב סטטיסטיקות
  distribution.statistics = this.calculateDistributionStats();

  return distribution;
}

generateFamilyScheduleView(weeklySchedule) {
  const familyView = {
    members: {},
```

```
  summary: {},
  recommendations: []
};

// ארגון לפי חבר משפחה
this.familyMembers.forEach(member => {
  familyView.members[member.id] = {
    name: member.name,
    role: member.role,
    tasks: [],
    totalHours: 0,
    dailyBreakdown: {}
  };
});

// מיון משימות לפי חבר משפחה
weeklySchedule.scheduledTasks
  .filter(item => item.assignedTo && item.assignedTo !== weeklySchedule.userId)
  .forEach(item => {
    const memberData = familyView.members[item.assignedTo];
    if (memberData) {
      memberData.tasks.push(item);
      memberData.totalHours += item.task.estimatedDuration / 60;

      const day = getDayName(item.day);
      memberData.dailyBreakdown[day] =
        (memberData.dailyBreakdown[day] || 0) + 1;
    }
  });

// חישוב סיכום
familyView.summary = {
  totalFamilyTasks: Object.values(familyView.members)
    .reduce((sum, m) => sum + m.tasks.length, 0),
  totalFamilyHours: Object.values(familyView.members)
    .reduce((sum, m) => sum + m.totalHours, 0),
  participation: {}
};

// חישוב אחוזי השתתפות
Object.entries(familyView.members).forEach(([id, data]) => {
  familyView.summary.participation[data.name] = {
    tasks: data.tasks.length,
    hours: data.totalHours,
    percentage: (data.tasks.length / familyView.summary.totalFamilyTasks) * 100
  };
});
```

```
  // המלצות
  familyView.recommendations = this.generateFamilyRecommendations(familyView);

  return familyView;
}

generateFamilyRecommendations(familyView) {
  const recommendations = [];

  // בדיקת איזון
  const participationRates = Object.values(familyView.summary.participation)
    .map(p => p.percentage);
  const stdDev = standardDeviation(participationRates);

  if (stdDev > 20) {
    recommendations.push({
      type: 'balance',
      priority: 'medium',
      message: 'יש פער גדול בחלוקת המשימות בין בני המשפחה. כדאי לאזן יותר.',
      action: 'rebalanceTasks'
    });
  }

  // בדיקת עומס יתר
  Object.entries(familyView.members).forEach(([id, data]) => {
    if (data.totalHours > 10) {
      recommendations.push({
        type: 'overload',
        priority: 'high',
        message: `${data.name} מתוכנן/ת ליותר מ-10 שעות השבוע. כדאי להפחית.`,
        action: 'reduceLoad',
        targetMember: id
      });
    }
  });

  // הצעות לשיתוף פעולה
  const collaborativeTasks = this.identifyCollaborativeOpportunities(familyView);
  if (collaborativeTasks.length > 0) {
    recommendations.push({
      type: 'collaboration',
      priority: 'low',
      message: 'יש משימות שאפשר לבצע ביחד כדי לחזק את הקשר המשפחתי.',
      tasks: collaborativeTasks
    });
  }
```

```
    return recommendations;
  }
};
```

2. מודל WeeklySchedule

javascript

```
const WeeklyScheduleSchema = new Schema({
  userId: { type: String, required: true },
  householdId: { type: String, required: true },
  weekStartDate: { type: Date, required: true },
  status: {
    type: String,
    enum: ['draft', 'active', 'archived'],
    default: 'draft'
  },

  // בלוקים קבועים (טיפולים, נסיעות, וכו')
  fixedBlocks: [{
    day: { type: Number, min: 0, max: 6 }, // 0=ראשון
    startTime: String, // "09:00"
    endTime: String, // "09:45"
    type: {
      type: String,
      enum: ['treatment', 'commute', 'personal', 'family']
    },
    location: String,
    assignedTo: String, // מי מהמשפחה
    recurring: { type: Boolean, default: false },
    notes: String
  }],

  // משימות משובצות
  scheduledTasks: [{
    taskId: { type: Schema.Types.ObjectId, ref: 'Task' },
    day: Number,
    startTime: String,
    endTime: String,
    location: String,
    assignedTo: { type: Schema.Types.ObjectId, ref: 'FamilyMember' },
    status: {
      type: String,
      enum: ['planned', 'in_progress', 'completed', 'postponed'],
      default: 'planned'
    },
    actualStartTime: String,
    actualEndTime: String,
    completionNotes: String,
    moodBeforeTask: { type: Number, min: 1, max: 10 },
    moodAfterTask: { type: Number, min: 1, max: 10 }
  }],

  // פרופיל אנרגיה צפוי
```

```
  energyProfile: [{
    day: Number,
    timeSlot: String, // 'morning', 'afternoon', 'evening'
    expectedEnergyLevel: String, // 'low', 'medium', 'high'
    notes: String // "אחרי טיפול - אנרגיה נמוכה"
  }],

  // מטא-דאטה
  metadata: {
    totalTasks: Number,
    totalFixedHours: Number,
    totalWorkHours: Number,
    familyParticipation: {
      teen16: { tasks: Number, hours: Number },
      teen14: { tasks: Number, hours: Number }
    }
  },

  createdAt: { type: Date, default: Date.now },
  lastModified: { type: Date, default: Date.now }
});
```

3. מודל FamilyMember

javascript

```
const FamilyMemberSchema = new Schema({
  householdId: { type: String, required: true },
  name: { type: String, required: true },
  role: {
    type: String,
    enum: ['parent', 'teen16', 'teen14'],
    required: true
  },
  age: Number,

  // זמינות
  availability: [{
    dayOfWeek: Number,
    timeSlots: [{
      start: String, // "14:00"
      end: String,   // "18:00"
      notes: String  // "אחרי בית ספר"
    }]
  }],

  // יכולות
  capabilities: [{
    type: String,
    enum: [
      'driving',          // נהיגה
      'cooking_simple',   // בישול פשוט
      'cooking_complex',  // בישול מורכב
      'shopping',         // קניות
      'companionship',    // ליווי
      'gardening',        // גינון
      'cleaning',         // ניקיון
      'laundry',          // כביסה
      'tech_help'         // עזרה טכנית
    ]
  }],

  // הגבלות
  maxTasksPerDay: { type: Number, default: 2 },
  maxHoursPerDay: { type: Number, default: 2 },
  schoolDays: [Number], // ימים בהם יש בית ספר

  // סטטיסטיקות
  stats: {
    tasksCompleted: { type: Number, default: 0 },
    averageCompletionRate: { type: Number, default: 100 },
    preferredTasks: [String]
```

```
  }
});
```

4. מודל DailyMoodCheck

javascript

```javascript
const DailyMoodCheckSchema = new Schema({
  userId: { type: String, required: true },
  date: { type: Date, required: true },

  // בדיקת בוקר
  morning: {
    time: Date,
    mood: { type: Number, min: 1, max: 10 },
    energy: { type: Number, min: 1, max: 10 },
    stress: { type: Number, min: 1, max: 10 },
    sleepQuality: { type: Number, min: 1, max: 10 },
    notes: String
  },

  // בדיקת ערב
  evening: {
    time: Date,
    mood: { type: Number, min: 1, max: 10 },
    energy: { type: Number, min: 1, max: 10 },
    stress: { type: Number, min: 1, max: 10 },
    accomplishmentFeeling: { type: Number, min: 1, max: 10 },
    notes: String
  },

  // אירועים מיוחדים
  significantEvents: [{
    time: Date,
    event: String,
    impact: { type: String, enum: ['positive', 'negative', 'neutral'] }
  }],

  // המלצות המערכת
  systemRecommendations: [{
    type: String,
    message: String,
    implemented: Boolean
  }]
});
```

ממשק משתמש - רכיבים מרכזיים

1. דף תכנון שבועי דינמי

jsx

```jsx
// WeeklyPlanner.jsx
function WeeklyPlanner() {
  const [schedule, setSchedule] = useState(null);
  const [treatments, setTreatments] = useState([]);
  const [familyAvailability, setFamilyAvailability] = useState({});
  const [viewMode, setViewMode] = useState('personal'); // 'personal', 'family', 'combined'

  return (
    <div className="weekly-planner">
     {/* כותרת עם אינדיקטורים */}
     <PlannerHeader>
       <WeekSelector
        onWeekChange={handleWeekChange}
        highlightThursday={true} // הדגשת יום חמישי
       />

       <QuickStats>
        <Stat icon="🎯" label="משימות" value={schedule?.totalTasks} />
        <Stat icon="⏰" label="שעות עבודה" value={schedule?.workHours} />
        <Stat icon="👨‍👩‍👧" label="משימות משפחה" value={schedule?.familyTasks} />
        <Stat icon="⚠️" label="דדליינים קרובים" value={urgentTasks.length} />
       </QuickStats>

       <ViewModeToggle
        mode={viewMode}
        onChange={setViewMode}
       />
     </PlannerHeader>

     {/* לוח זמנים ראשי */}
     <div className="planner-body">
       {/* פאנל צד - משימות ממתינות */}
       <TasksSidebar>
        <TaskFilters>
          <FilterButton icon="🧠" label="לפי עומס קוגניטיבי" />
          <FilterButton icon="📍" label="לפי מיקום" />
          <FilterButton icon="⏱️" label="לפי משך" />
          <FilterButton icon="👨‍👩‍👧" label="מתאים למשפחה" />
        </TaskFilters>

        <PendingTasksList
          tasks={pendingTasks}
          onDragStart={handleDragStart}
        />

        <QuickAddTask>
```

```
    <input
      placeholder="הוסף משימה מהירה..."
      onKeyPress={handleQuickAdd}
    />
  </QuickAddTask>
</TasksSidebar>

{/* גריד שבועי */}
<WeeklyGrid>
  <TimeColumn>
    {generateTimeSlots().map(time => (
      <TimeSlot key={time}>{time}</TimeSlot>
    ))}
  </TimeColumn>

  {days.map(day => (
    <DayColumn key={day.number}>
      <DayHeader>
        <span>{day.name}</span>
        {day.number === 4 && <Badge>יום טיפולים</Badge>}
      </DayHeader>

      {/* בלוקים קבועים */}
      {getFixedBlocks(day.number).map(block => (
        <FixedBlock
          key={block.id}
          {...block}
          className={block.type}
        />
      ))}

      {/* משימות משובצות */}
      {getScheduledTasks(day.number).map(task => (
        <ScheduledTask
          key={task.id}
          {...task}
          draggable={true}
          onDrop={handleTaskDrop}
          onClick={() => showTaskDetails(task)}
        />
      ))}

      {/* חלונות זמן פנויים */}
      {getAvailableSlots(day.number).map(slot => (
        <AvailableSlot
          key={slot.id}
          {...slot}
```

```
              onDrop={handleTaskDrop}
              onClick={() => showSuggestions(slot)}
            />
          ))}
        </DayColumn>
      ))}
    </WeeklyGrid>
  </div>

  {/* פאנל תחתון - כלים מהירים */}
  <PlannerFooter>
    <ActionButton
      icon="🤖"
      label="תכנון אוטומטי חכם"
      onClick={runSmartScheduling}
      primary
    />
    <ActionButton
      icon="👪"
      label="הצע משימות למשפחה"
      onClick={suggestFamilyTasks}
    />
    <ActionButton
      icon="📊"
      label="ניתוח עומסים"
      onClick={showWorkloadAnalysis}
    />
    <ActionButton
      icon="📅"
      label="ייצוא ללוח שנה"
      onClick={exportToCalendar}
    />
  </PlannerFooter>
</div>
);
}
```

2. ממשק הזנת טיפולים (Thursday-centric)

jsx

```jsx
// TreatmentScheduleInput.jsx
function TreatmentScheduleInput() {
  const [inputMethod, setInputMethod] = useState('manual'); // 'manual', 'photo', 'import'
  const [treatments, setTreatments] = useState([]);
  const [companionSchedule, setCompanionSchedule] = useState({});

  return (
    <div className="treatment-input">
      <div className="header">
        <h2>עדכון לוח טיפולים שבועי</h2>
        <p className="subtitle">מתעדכן כל יום חמישי בצהריים</p>
      </div>

      {/* בחירת שיטת הזנה */}
      <InputMethodTabs>
        <Tab
          active={inputMethod === 'manual'}
          onClick={() => setInputMethod('manual')}
        >
          הזנה ידנית
        </Tab>
        <Tab
          active={inputMethod === 'photo'}
          onClick={() => setInputMethod('photo')}
        >
          צילום לוח זמנים
        </Tab>
        <Tab
          active={inputMethod === 'import'}
          onClick={() => setInputMethod('import')}
        >
          ייבוא מקובץ
        </Tab>
      </InputMethodTabs>

      {/* הזנה ידנית */}
      {inputMethod === 'manual' && (
        <ManualInput>
          {['ראשון', 'שני', 'שלישי', 'רביעי', 'חמישי'].map((day, index) => (
            <DaySection key={day}>
              <h3>{day}</h3>
              <TreatmentSlots>
                {treatments
                  .filter(t => t.day === index)
                  .map((treatment, idx) => (
                    <TreatmentSlot key={idx}>
```

```
                <TimeInput
                  value={treatment.startTime}
                  onChange={(time) => updateTreatment(treatment.id, { startTime: time })}
                />
                <span>-</span>
                <TimeInput
                  value={treatment.endTime}
                  onChange={(time) => updateTreatment(treatment.id, { endTime: time })}
                />
                <Select
                  value={treatment.companion}
                  onChange={(companion) => updateTreatment(treatment.id, { companion })}
                >
                  <option value="">ללא ליווי</option>
                  <option value="spouse">אני</option>
                  <option value="daughter16">הבת הגדולה</option>
                </Select>
                <DeleteButton onClick={() => removeTreatment(treatment.id)} />
              </TreatmentSlot>
            ))}
            <AddButton onClick={() => addTreatment(index)}>
              + הוסף טיפול
            </AddButton>
          </TreatmentSlots>
        </DaySection>
      ))}
    </ManualInput>
  )}

  {/* העלאת תמונה */}
  {inputMethod === 'photo' && (
    <PhotoUpload>
      <DropZone
        accept="image/*"
        onDrop={handlePhotoUpload}
      >
        <CameraIcon />
        <p>גרור תמונה או לחץ לבחירה</p>
      </DropZone>
      {uploadedPhoto && (
        <PhotoPreview>
          <img src={uploadedPhoto} alt="לוח טיפולים" />
          <ProcessButton onClick={processPhoto}>
            עבד תמונה וחלץ זמנים
          </ProcessButton>
        </PhotoPreview>
      )}
```

```
        </PhotoUpload>
      )}

      {/* סיכום וחישוב השפעות */}
      <ScheduleSummary>
        <h3>סיכום שבועי</h3>
        <SummaryGrid>
          <SummaryItem>
            <label>סה"כ טיפולים:</label>
            <value>{treatments.length}</value>
          </SummaryItem>
          <SummaryItem>
            <label>זמן בטיפולים:</label>
            <value>{calculateTotalTreatmentTime()} שעות</value>
          </SummaryItem>
          <SummaryItem>
            <label>זמן נסיעות:</label>
            <value>{calculateCommuteTime()} שעות</value>
          </SummaryItem>
          <SummaryItem>
            <label>חלונות עבודה במשרד:</label>
            <value>{calculateOfficeWindows()} חלונות</value>
          </SummaryItem>
        </SummaryGrid>

        <Recommendations>
          <h4>המלצות:</h4>
          <ul>
            {generateRecommendations().map((rec, idx) => (
              <li key={idx}>{rec}</li>
            ))}
          </ul>
        </Recommendations>
      </ScheduleSummary>

      <ActionButtons>
        <Button variant="secondary" onClick={saveDraft}>
          שמור כטיוטה
        </Button>
        <Button variant="primary" onClick={generateSchedule}>
          צור תכנון שבועי חכם
        </Button>
      </ActionButtons>
    </div>
  );
}
```

3. לוח בקרה משפחתי

jsx

```jsx
// FamilyDashboard.jsx
function FamilyDashboard() {
  const [familyMembers, setFamilyMembers] = useState([]);
  const [weeklyTasks, setWeeklyTasks] = useState([]);
  const [showApprovalQueue, setShowApprovalQueue] = useState(false);

  return (
    <div className="family-dashboard">
      <h2>מרכז המשפחה</h2>

      {/* כרטיסי בני משפחה */}
      <FamilyCards>
        {familyMembers.map(member => (
          <MemberCard key={member.id}>
            <MemberHeader>
              <Avatar>{member.name[0]}</Avatar>
              <MemberInfo>
                <h3>{member.name}</h3>
                <p>{member.age} שנים</p>
              </MemberInfo>
            </MemberHeader>

            <MemberStats>
              <Stat>
                <label>משימות היום</label>
                <value>{member.todayTasks.length}/{member.maxTasksPerDay}</value>
              </Stat>
              <Stat>
                <label>זמן מושקע</label>
                <value>{member.hoursToday} שעות</value>
              </Stat>
            </MemberStats>

            <CurrentTasks>
              {member.currentTasks.map(task => (
                <TaskChip key={task.id} status={task.status}>
                  {task.name}
                </TaskChip>
              ))}
            </CurrentTasks>

            <Capabilities>
              {member.capabilities.map(cap => (
                <CapabilityBadge key={cap}>
                  {getCapabilityIcon(cap)} {getCapabilityName(cap)}
                </CapabilityBadge>
```

```
        ))}
      </Capabilities>
    </MemberCard>
  ))}
</FamilyCards>

{/* משימות משפחתיות לשבוע */}
<WeeklyFamilyTasks>
  <h3>משימות משפחתיות השבוע</h3>
  <TasksGrid>
    {weeklyTasks.map(task => (
      <FamilyTask key={task.id}>
        <TaskHeader>
          <TaskName>{task.name}</TaskName>
          <TaskBadges>
            {task.urgent && <UrgentBadge />}
            {task.recurring && <RecurringBadge />}
          </TaskBadges>
        </TaskHeader>

        <TaskDetails>
          <Detail icon="📅">{formatDate(task.dueDate)}</Detail>
          <Detail icon="⏱️">{task.estimatedDuration} דקות</Detail>
          <Detail icon="📍">{task.location}</Detail>
        </TaskDetails>

        <AssignmentSection>
          {task.assignedTo ? (
            <AssignedTo>
              <Avatar small>{task.assignedTo.name[0]}</Avatar>
              <span>{task.assignedTo.name}</span>
              {task.parentApprovalRequired && !task.approved && (
                <ApprovalButton onClick={() => approveTask(task.id)}>
                  אשר
                </ApprovalButton>
              )}
            </AssignedTo>
          ) : (
            <AssignButton onClick={() => showAssignmentOptions(task)}>
              הקצה למישהו
            </AssignButton>
          )}
        </AssignmentSection>
      </FamilyTask>
    ))}
  </TasksGrid>
</WeeklyFamilyTasks>
```

```
      {/* תור אישורים */}
      {showApprovalQueue && (
        <ApprovalQueue>
          <h3>ממתין לאישור</h3>
          {pendingApprovals.map(item => (
            <ApprovalItem key={item.id}>
              <ItemInfo>
                <p><strong>{item.memberName}</strong> רוצה לקחת:</p>
                <p>{item.taskName}</p>
              </ItemInfo>
              <ApprovalActions>
                <Button variant="success" onClick={() => approve(item.id)}>
                  אשר
                </Button>
                <Button variant="danger" onClick={() => reject(item.id)}>
                  דחה
                </Button>
              </ApprovalActions>
            </ApprovalItem>
          ))}
        </ApprovalQueue>
      )}

      {/* כללי בית וטיפים */}
      <HouseRules>
        <h3>כללי הבית למשימות</h3>
        <Rules>
          <Rule>📚 לימודים לפני משימות בית</Rule>
          <Rule>🎮 מקסימום 2 משימות ביום לכל ילד</Rule>
          <Rule>💰 בונוס על משימות מעבר למכסה</Rule>
          <Rule>🤝 עזרה הדדית מעודדת</Rule>
        </Rules>
      </HouseRules>
    </div>
  );
}
```

4. מעקב מצב רוח יומי

jsx

```jsx
// DailyMoodTracker.jsx
function DailyMoodTracker() {
  const [checkIn, setCheckIn] = useState({
    mood: 5,
    energy: 5,
    stress: 5,
    sleep: 5
  });
  const [showHistory, setShowHistory] = useState(false);
  const [insights, setInsights] = useState(null);

  return (
    <div className="mood-tracker">
     {/* בדיקה מהירה */}
     <QuickCheckIn>
       <h3>איך את מרגישה עכשיו?</h3>
       <p className="time">{new Date().toLocaleTimeString('he-IL')}</p>

       <CheckInSliders>
         <SliderItem>
           <label>
             <span> 😔 </span>
             <span>מצב רוח</span>
             <span> 😊 </span>
           </label>
           <Slider
             min={1}
             max={10}
             value={checkIn.mood}
             onChange={(value) => updateCheckIn('mood', value)}
             color={getMoodColor(checkIn.mood)}
           />
           <value>{checkIn.mood}/10</value>
         </SliderItem>

         <SliderItem>
           <label>
             <span> 🔋 </span>
             <span>אנרגיה</span>
             <span> ⚡ </span>
           </label>
           <Slider
             min={1}
             max={10}
             value={checkIn.energy}
             onChange={(value) => updateCheckIn('energy', value)}
```

```
      color={getEnergyColor(checkIn.energy)}
    />
    <value>{checkIn.energy}/10</value>
  </SliderItem>

  <SliderItem>
    <label>
      <span>😔</span>
      <span>רמת לחץ</span>
      <span>😰</span>
    </label>
    <Slider
      min={1}
      max={10}
      value={checkIn.stress}
      onChange={(value) => updateCheckIn('stress', value)}
      color={getStressColor(checkIn.stress)}
    />
    <value>{checkIn.stress}/10</value>
  </SliderItem>

  <SliderItem>
    <label>
      <span>😴</span>
      <span>איכות שינה</span>
      <span>🛏️</span>
    </label>
    <Slider
      min={1}
      max={10}
      value={checkIn.sleep}
      onChange={(value) => updateCheckIn('sleep', value)}
      color={getSleepColor(checkIn.sleep)}
    />
    <value>{checkIn.sleep}/10</value>
  </SliderItem>
</CheckInSliders>

<QuickNotes>
  <input
    type="text"
    placeholder="הערה קצרה (אופציונלי)..."
    maxLength={100}
  />
</QuickNotes>

<SaveButton onClick={saveCheckIn}>
```

```
      שמור (5 שניות) ✨
    </SaveButton>
  </QuickCheckIn>

  {/* תובנות מיידיות */}
  {insights && (
    <InstantInsights>
      <h4>תובנות</h4>
      {insights.recommendations.map((rec, idx) => (
        <Insight key={idx} type={rec.type}>
          <InsightIcon>{rec.icon}</InsightIcon>
          <InsightText>{rec.text}</InsightText>
          {rec.actionable && (
            <ActionLink onClick={() => handleInsightAction(rec)}>
              {rec.actionText}
            </ActionLink>
          )}
        </Insight>
      ))}
    </InstantInsights>
  )}

  {/* היסטוריה וגרפים */}
  <HistoryToggle onClick={() => setShowHistory(!showHistory)}>
    {showHistory ? 'הסתר' : 'הצג'} היסטוריה
  </HistoryToggle>

  {showHistory && (
    <MoodHistory>
      <TimeRangeSelector>
        <option value="week">שבוע אחרון</option>
        <option value="month">חודש אחרון</option>
        <option value="custom">טווח מותאם</option>
      </TimeRangeSelector>

      <MoodChart>
        {/* גרף קווים פשוט */}
        <LineChart data={moodHistory} />
      </MoodChart>

      <Patterns>
        <h4>דפוסים שזיהינו:</h4>
        <PatternList>
          {patterns.map((pattern, idx) => (
            <Pattern key={idx}>
              <PatternIcon>{pattern.icon}</PatternIcon>
              <PatternText>
```

```
                <strong>{pattern.title}</strong>
                <p>{pattern.description}</p>
              </PatternText>
            </Pattern>
          ))}
        </PatternList>
      </Patterns>
    </MoodHistory>
  )}
</div>
);
}
```

לוגיקה עסקית ואלגוריתמים

1. מנוע התכנון החכם

javascript

```javascript
// SmartSchedulingEngine.js
class SmartSchedulingEngine {
  constructor(userId, weekStart) {
    this.userId = userId;
    this.weekStart = weekStart;
    this.userPreferences = null;
    this.familyMembers = null;
    this.historicalData = null;
    this.moodPatterns = null;
  }

  async generateOptimalSchedule() {
    // טעינת כל הנתונים הרלוונטיים
    await this.loadAllData();

    // שלב 1: ניתוח מגבלות וזמינות
    const constraints = await this.analyzeConstraints();

    // שלב 2: דירוג וסיווג משימות
    const prioritizedTasks = await this.prioritizeAndCategorizeTasks();

    // שלב 3: חישוב חלונות זמן אופטימליים
    const timeSlots = this.calculateOptimalTimeSlots(constraints);

    // שלב 4: התאמת משימות לחלונות זמן
    const schedule = this.matchTasksToSlots(prioritizedTasks, timeSlots);

    // שלב 5: חלוקת משימות משפחתיות
    const familySchedule = await this.distributeFamilyTasks(schedule);

    // שלב 6: אופטימיזציה סופית
    const optimized = this.optimizeSchedule(familySchedule);

    // שלב 7: בדיקת כללים ואילוצים
    const validated = await this.validateSchedule(optimized);

    return validated;
  }

  async analyzeConstraints() {
    const constraints = {
      fixed: [],
      flexible: [],
      forbidden: []
    };
```

```
  // טיפולים וזמני נסיעה
  const treatments = await this.getTreatmentSchedule();
  treatments.forEach(treatment => {
    // הטיפול עצמו
    constraints.fixed.push({
      day: treatment.day,
      start: treatment.startTime,
      end: treatment.endTime,
      type: 'treatment',
      priority: 'absolute'
    });

    // זמן נסיעה לפני
    constraints.fixed.push({
      day: treatment.day,
      start: subtractMinutes(treatment.startTime, 60),
      end: treatment.startTime,
      type: 'commute',
      priority: 'absolute'
    });

    // זמן נסיעה אחרי
    constraints.fixed.push({
      day: treatment.day,
      start: treatment.endTime,
      end: addMinutes(treatment.endTime, 60),
      type: 'commute',
      priority: 'absolute'
    });
  });

  // זמני משפחה קבועים
  const familyTime = await this.getFamilyConstraints();
  constraints.flexible.push(...familyTime);

  // אזורי "אסור" - זמני מנוחה הכרחיים
  const restPeriods = this.calculateRequiredRestPeriods();
  constraints.forbidden.push(...restPeriods);

  return constraints;
}

prioritizeAndCategorizeTasks() {
  return this.tasks.map(task => {
    const score = this.calculateTaskPriority(task);
    const category = this.categorizeTask(task);
    const familySuitability = this.assessFamilySuitability(task);
```

```
    return {
      ...task,
      priorityScore: score,
      category,
      familySuitability,
      schedulingPreferences: this.getTaskPreferences(task)
    };
  }).sort((a, b) => b.priorityScore - a.priorityScore);
}

calculateTaskPriority(task) {
  let score = 0;

  // דחיפות לפי דדליין
  const daysUntilDeadline = differenceInDays(task.deadline, new Date());
  if (daysUntilDeadline <= 0) {
    score += 1000; // דחוף ביותר
  } else if (daysUntilDeadline <= task.bufferDays) {
    score += 500 / daysUntilDeadline;
  } else {
    score += 100 / daysUntilDeadline;
  }

  // חשיבות עסקית
  const businessImportance = {
    'salary': 200,
    'vat': 180,
    'reconciliation': 150,
    'reporting': 120,
    'client_urgent': 160,
    'development': 80,
    'marketing': 70,
    'admin': 50
  };
  score += businessImportance[task.category] || 30;

  // השפעת תלויות
  if (task.dependencies?.length > 0) {
    const uncompletedDeps = task.dependencies.filter(
      depId => !this.isTaskCompleted(depId)
    );
    score -= uncompletedDeps.length * 50;
  }

  // התחשבות בהיסטוריה
  const historicalSuccess = this.getHistoricalSuccessRate(task.type);
```

```
  score += historicalSuccess * 50;

  // התאמה למצב נוכחי
  const currentState = this.getCurrentUserState();
  if (task.cognitiveLoad === 'high' && currentState.energy < 5) {
    score -= 100; // לא מתאים כרגע
  }

  return Math.max(score, 0);
}

calculateOptimalTimeSlots(constraints) {
  const slots = [];
  const workingHours = this.getUserWorkingHours();

  for (let day = 0; day < 7; day++) {
    const daySlots = [];
    let currentTime = workingHours.start;

    while (currentTime < workingHours.end) {
      const slot = {
        day,
        start: currentTime,
        end: addMinutes(currentTime, 30),
        quality: this.assessSlotQuality(day, currentTime),
        constraints: []
      };

      // בדוק התנגשויות עם אילוצים
      const conflicts = this.findConflicts(slot, constraints);
      if (conflicts.length === 0) {
        slot.available = true;
      } else {
        slot.available = false;
        slot.constraints = conflicts;
      }

      // הערך איכות לפי היסטוריה ומצב צפוי
      slot.energyLevel = this.predictEnergyLevel(day, currentTime);
      slot.productivityScore = this.predictProductivity(day, currentTime);

      daySlots.push(slot);
      currentTime = addMinutes(currentTime, 30);
    }

    slots.push(...daySlots);
  }
```

```
  return slots;
}

matchTasksToSlots(tasks, slots) {
  const schedule = [];
  const remainingSlots = [...slots.filter(s => s.available)];
  const unscheduledTasks = [];

  for (const task of tasks) {
    const bestMatch = this.findBestSlotForTask(task, remainingSlots);

    if (bestMatch) {
      // שיבוץ המשימה
      const scheduledItem = {
        taskId: task.id,
        task: task,
        day: bestMatch.slot.day,
        startTime: bestMatch.slot.start,
        endTime: addMinutes(bestMatch.slot.start, task.estimatedDuration),
        location: this.determineLocation(task, bestMatch.slot),
        assignedTo: task.suitableFor.includes('parent') ? this.userId : null,
        confidence: bestMatch.score,
        reason: bestMatch.reason
      };

      schedule.push(scheduledItem);

      // עדכון חלונות זמן פנויים
      this.updateAvailableSlots(remainingSlots, bestMatch.slot, task);
    } else {
      unscheduledTasks.push(task);
    }
  }

  // נסה לשבץ משימות שנותרו בחלונות פחות אופטימליים
  for (const task of unscheduledTasks) {
    const fallbackSlot = this.findFallbackSlot(task, remainingSlots);
    if (fallbackSlot) {
      schedule.push({
        taskId: task.id,
        task: task,
        ...fallbackSlot,
        confidence: 'low',
        reason: 'fallback'
      });
    }
```

```
  }

  return schedule;
}

findBestSlotForTask(task, availableSlots) {
  let bestMatch = null;
  let bestScore = -1;

  for (const slot of availableSlots) {
    // בדוק אם המשימה מתאימה לחלון
    if (!this.canTaskFitInSlot(task, slot)) continue;

    const matchScore = this.calculateMatchScore(task, slot);

    if (matchScore.total > bestScore) {
      bestScore = matchScore.total;
      bestMatch = {
        slot,
        score: matchScore.total,
        breakdown: matchScore,
        reason: this.explainMatch(task, slot, matchScore)
      };
    }
  }

  return bestMatch;
}

calculateMatchScore(task, slot) {
  const scores = {
    energy: 0,
    location: 0,
    timing: 0,
    productivity: 0,
    preference: 0
  };

  // התאמת אנרגיה
  if (task.energyLevel === 'high' && slot.energyLevel >= 7) {
    scores.energy = 30;
  } else if (task.energyLevel === 'medium' && slot.energyLevel >= 5) {
    scores.energy = 25;
  } else if (task.energyLevel === 'low' && slot.energyLevel >= 3) {
    scores.energy = 20;
  } else {
    scores.energy = Math.max(0, 10 - Math.abs(task.energyLevel - slot.energyLevel) * 2);
```

```
  }

  // התאמת מיקום
  const slotLocation = this.getSlotLocation(slot);
  if (task.locationFlexibility === 'anywhere') {
    scores.location = 20;
  } else if (task.locationFlexibility === 'remote_possible' && slotLocation !== 'office') {
    scores.location = 25;
  } else if (task.locationFlexibility === 'office_only' && slotLocation === 'office') {
    scores.location = 30;
  } else if (task.locationFlexibility === 'mobile_friendly' && slotLocation === 'transit') {
    scores.location = 15;
  }

  // התאמת זמן ביום
  const timeOfDay = this.getTimeOfDay(slot.start);
  if (task.preferredTimeOfDay === 'anytime') {
    scores.timing = 15;
  } else if (task.preferredTimeOfDay === timeOfDay) {
    scores.timing = 30;
  } else {
    scores.timing = 5;
  }

  // פרודוקטיביות צפויה
  scores.productivity = slot.productivityScore * 0.3;

  // העדפות משתמש
  const userPref = this.getUserPreferenceForTaskType(task.type, slot);
  scores.preference = userPref * 0.2;

  // חישוב ציון כולל
  scores.total = Object.values(scores).reduce((sum, score) => sum + score, 0);

  return scores;
}

async distributeFamilyTasks(schedule) {
  const familyTasks = schedule.filter(
    item => item.task.suitableFor.some(f => f !== 'parent')
  );

  const balancer = new FamilyLoadBalancer(this.familyMembers);

  for (const item of familyTasks) {
    if (item.assignedTo) continue; // כבר משובץ
```

```
      const suggestions = await balancer.suggestAssignment(item.task, item);

      if (suggestions.length > 0) {
        // בחר את ההתאמה הטובה ביותר
        const bestMatch = suggestions[0];

        // בדוק אם דרוש אישור הורה
        if (item.task.parentApprovalRequired) {
          item.assignedTo = bestMatch.member.id;
          item.status = 'pending_approval';
          item.assignmentScore = bestMatch.score;
        } else {
          item.assignedTo = bestMatch.member.id;
          item.status = 'assigned';
        }
      }
    }

    return schedule;
  }

  optimizeSchedule(schedule) {
    let optimized = [...schedule];
    let improved = true;
    let iterations = 0;
    const maxIterations = 10;

    while (improved && iterations < maxIterations) {
      improved = false;
      iterations++;

      // נסה להפחית "חורים" בלוח הזמנים
      const gaps = this.findScheduleGaps(optimized);
      for (const gap of gaps) {
        const filled = this.tryFillGap(gap, optimized);
        if (filled) {
          optimized = filled;
          improved = true;
        }
      }

      // נסה לקבץ משימות דומות
      const grouped = this.tryGroupSimilarTasks(optimized);
      if (grouped.length !== optimized.length) {
        optimized = grouped;
        improved = true;
      }
```

```
    // אזן עומסים בין ימים
    const balanced = this.balanceDailyLoad(optimized);
    if (this.calculateScheduleScore(balanced) > this.calculateScheduleScore(optimized)) {
      optimized = balanced;
      improved = true;
    }
  }

  return optimized;
}

async validateSchedule(schedule) {
  const validationResults = {
    valid: true,
    warnings: [],
    errors: [],
    suggestions: []
  };

  // בדיקת התנגשויות
  const conflicts = this.findTimeConflicts(schedule);
  if (conflicts.length > 0) {
    validationResults.valid = false;
    validationResults.errors.push(...conflicts.map(c => ({
      type: 'conflict',
      message: `התנגשות בין ${c.task1.name} ל-${c.task2.name}`,
      items: [c.task1.id, c.task2.id]
    })));
  }

  // בדיקת עומס יתר
  const overloadedDays = this.findOverloadedDays(schedule);
  overloadedDays.forEach(day => {
    validationResults.warnings.push({
      type: 'overload',
      message: `יום ${day.name} עמוס מדי (${day.hours} שעות)`,
      severity: day.hours > 10 ? 'high' : 'medium'
    });
  });

  // בדיקת דדליינים
  const missedDeadlines = this.checkDeadlines(schedule);
  missedDeadlines.forEach(task => {
    validationResults.errors.push({
      type: 'deadline',
      message: `${task.name} לא ישובץ לפני הדדליין`,
```

```
        taskId: task.id
      });
    });

    // בדיקת איזון משפחתי
    const familyBalance = this.checkFamilyBalance(schedule);
    if (!familyBalance.balanced) {
      validationResults.warnings.push({
        type: 'family_balance',
        message: familyBalance.message,
        suggestions: familyBalance.suggestions
      });
    }

    // הצעות לשיפור
    const improvements = await this.suggestImprovements(schedule);
    validationResults.suggestions.push(...improvements);

    return {
      schedule,
      validation: validationResults
    };
  }
}
```

2. מנוע למידה והתאמה אישית

javascript

```
// PersonalizationEngine.js
class PersonalizationEngine {
  constructor(userId) {
    this.userId = userId;
    this.learningData = {
      taskCompletion: [],
      timeEstimates: [],
      energyPatterns: [],
      productivityPeriods: [],
      stressIndicators: []
    };
  }

  async learnFromHistory(days = 30) {
    // טען היסטוריית ביצועים
    const history = await this.loadExecutionHistory(days);

    // ניתוח דפוסי השלמת משימות
    this.analyzeCompletionPatterns(history);

    // ניתוח דיוק בהערכות זמן
    this.analyzeTimeEstimateAccuracy(history);

    // ניתוח רמות אנרגיה לאורך היום
    this.analyzeEnergyPatterns(history);

    // זיהוי תקופות פרודוקטיביות
    this.identifyProductivePeriods(history);

    // זיהוי טריגרים ללחץ
    this.identifyStressTriggers(history);

    // בניית פרופיל משתמש מעודכן
    return this.buildUserProfile();
  }

  analyzeCompletionPatterns(history) {
    const patterns = {};

    // ניתוח לפי סוג משימה
    history.forEach(record => {
      const taskType = record.task.type;
      if (!patterns[taskType]) {
        patterns[taskType] = {
          total: 0,
          completed: 0,
```

```
        onTime: 0,
        delayed: 0,
        postponed: 0,
        averageDelay: []
      };
    }

    patterns[taskType].total++;

    if (record.status === 'completed') {
      patterns[taskType].completed++;

      if (record.completedOnTime) {
        patterns[taskType].onTime++;
      } else {
        patterns[taskType].delayed++;
        const delay = differenceInMinutes(
          record.actualEndTime,
          record.plannedEndTime
        );
        patterns[taskType].averageDelay.push(delay);
      }
    } else if (record.status === 'postponed') {
      patterns[taskType].postponed++;
    }
  });

  // חישוב ממוצעים ואחוזים
  Object.keys(patterns).forEach(type => {
    const p = patterns[type];
    p.completionRate = (p.completed / p.total) * 100;
    p.onTimeRate = (p.onTime / p.completed) * 100 || 0;
    p.averageDelayMinutes = p.averageDelay.length > 0
      ? average(p.averageDelay)
      : 0;
  });

  this.learningData.taskCompletion = patterns;
}

analyzeTimeEstimateAccuracy(history) {
  const accuracy = {};

  history.filter(r => r.actualDuration).forEach(record => {
    const taskType = record.task.type;
    const cognitiveLoad = record.task.cognitiveLoad;
    const key = `${taskType}_${cognitiveLoad}`;
```

```
      if (!accuracy[key]) {
        accuracy[key] = {
          estimates: [],
          actuals: [],
          ratios: []
        };
      }

      accuracy[key].estimates.push(record.task.estimatedDuration);
      accuracy[key].actuals.push(record.actualDuration);
      accuracy[key].ratios.push(
        record.actualDuration / record.task.estimatedDuration
      );
    });

    // חישוב מקדמי תיקון
    Object.keys(accuracy).forEach(key => {
      const data = accuracy[key];
      data.averageRatio = average(data.ratios);
      data.standardDeviation = standardDeviation(data.ratios);
      data.correctionFactor = data.averageRatio;

      // המלצה להתאמת הערכות
      if (data.averageRatio > 1.2) {
        data.recommendation = 'increase_estimates';
        data.suggestedMultiplier = Math.min(data.averageRatio, 1.5);
      } else if (data.averageRatio < 0.8) {
        data.recommendation = 'decrease_estimates';
        data.suggestedMultiplier = Math.max(data.averageRatio, 0.7);
      } else {
        data.recommendation = 'maintain_estimates';
        data.suggestedMultiplier = 1;
      }
    });

    this.learningData.timeEstimates = accuracy;
  }

  analyzeEnergyPatterns(history) {
    const patterns = {
      hourly: {},
      daily: {},
      taskType: {}
    };

    // ניתוח לפי שעות ביום
```

```
  for (let hour = 0; hour < 24; hour++) {
    patterns.hourly[hour] = {
      moodReadings: [],
      energyReadings: [],
      taskSuccess: [],
      stressLevels: []
    };
  }

  history.forEach(record => {
    const hour = new Date(record.startTime).getHours();

    if (record.moodBeforeTask) {
      patterns.hourly[hour].moodReadings.push(record.moodBeforeTask);
    }
    if (record.energyLevel) {
      patterns.hourly[hour].energyReadings.push(record.energyLevel);
    }
    if (record.status === 'completed') {
      patterns.hourly[hour].taskSuccess.push(
        record.completedOnTime ? 1 : 0.5
      );
    }
    if (record.stressLevel) {
      patterns.hourly[hour].stressLevels.push(record.stressLevel);
    }
  });

  // חישוב ממוצעים לכל שעה
  Object.keys(patterns.hourly).forEach(hour => {
    const data = patterns.hourly[hour];
    data.averageMood = average(data.moodReadings) || 5;
    data.averageEnergy = average(data.energyReadings) || 5;
    data.successRate = average(data.taskSuccess) || 0.5;
    data.averageStress = average(data.stressLevels) || 5;

    // דירוג איכות השעה
    data.qualityScore =
      (data.averageMood * 0.2) +
      (data.averageEnergy * 0.3) +
      (data.successRate * 50 * 0.3) +
      ((10 - data.averageStress) * 0.2);
  });

  this.learningData.energyPatterns = patterns;
}
```

```
identifyProductivePeriods(history) {
  const periods = [];
  const hourlyData = this.learningData.energyPatterns.hourly;

  // מצא רצפים של שעות פרודוקטיביות
  let currentPeriod = null;

  for (let hour = 0; hour < 24; hour++) {
    const quality = hourlyData[hour].qualityScore;

    if (quality >= 7) { // סף לשעה פרודוקטיבית
      if (!currentPeriod) {
        currentPeriod = {
          start: hour,
          end: hour,
          averageQuality: quality,
          hours: [hour]
        };
      } else {
        currentPeriod.end = hour;
        currentPeriod.hours.push(hour);
        currentPeriod.averageQuality =
          average(currentPeriod.hours.map(h => hourlyData[h].qualityScore));
      }
    } else if (currentPeriod) {
      if (currentPeriod.hours.length >= 2) { // לפחות 2 שעות רצופות
        periods.push(currentPeriod);
      }
      currentPeriod = null;
    }
  }

  // סיום תקופה אחרונה
  if (currentPeriod && currentPeriod.hours.length >= 2) {
    periods.push(currentPeriod);
  }

  // מיון לפי איכות ממוצעת
  periods.sort((a, b) => b.averageQuality - a.averageQuality);

  this.learningData.productivityPeriods = periods.map(period => ({
    ...period,
    name: this.namePeriod(period),
    recommendations: this.generatePeriodRecommendations(period)
  }));
}
```

```
identifyStressTriggers(history) {
  const triggers = {
    taskTypes: {},
    timeConstraints: {},
    workload: {},
    patterns: []
  };

  // ניתוח לפי סוג משימה
  history.forEach(record => {
    if (record.stressLevel >= 7) { // רמת לחץ גבוהה
      // סוג משימה
      const taskType = record.task.type;
      triggers.taskTypes[taskType] =
        (triggers.taskTypes[taskType] || 0) + 1;

      // אילוצי זמן
      const timeToDeadline = differenceInHours(
        record.task.deadline,
        record.startTime
      );
      if (timeToDeadline < 24) {
        triggers.timeConstraints['last_minute'] =
          (triggers.timeConstraints['last_minute'] || 0) + 1;
      }

      // עומס יומי
      const dailyTasks = this.getTasksForDay(
        history,
        new Date(record.startTime)
      );
      if (dailyTasks.length > 8) {
        triggers.workload['overloaded'] =
          (triggers.workload['overloaded'] || 0) + 1;
      }
    }
  });

  // זיהוי דפוסים
  triggers.patterns = this.extractStressPatterns(triggers);

  this.learningData.stressIndicators = triggers;
}

buildUserProfile() {
  return {
    userId: this.userId,
```

```javascript
      learningData: this.learningData,
      preferences: {
        optimalWorkingHours: this.determineOptimalHours(),
        taskTypePreferences: this.determineTaskPreferences(),
        energyManagement: this.determineEnergyStrategy(),
        stressAvoidance: this.determineStressAvoidance()
      },
      recommendations: this.generatePersonalizedRecommendations()
    };
  }

  generatePersonalizedRecommendations() {
    const recommendations = [];

    // המלצות על סמך דפוסי אנרגיה
    const topPeriods = this.learningData.productivityPeriods.slice(0, 3);
    if (topPeriods.length > 0) {
      recommendations.push({
        type: 'scheduling',
        priority: 'high',
        message: `התקופות הפרודוקטיביות ביותר שלך הן: ${topPeriods.map(p => p.name).join(', ')}. ות מורכבות לשעות אלו
        actionable: true,
        action: 'optimizeScheduleForProductivity'
      });
    }

    // המלצות על סמך דיוק הערכות זמן
    const overestimated = Object.entries(this.learningData.timeEstimates)
      .filter(([_, data]) => data.averageRatio < 0.8);
    const underestimated = Object.entries(this.learningData.timeEstimates)
      .filter(([_, data]) => data.averageRatio > 1.2);

    if (underestimated.length > 0) {
      recommendations.push({
        type: 'estimation',
        priority: 'medium',
        message: `נראה שאת נוטה להעריך פחות מדי זמן למשימות מסוג ${underestimated.map(([type]) => type.split('_')[0]).
        actionable: true,
        action: 'adjustTimeEstimates'
      });
    }

    // המלצות למניעת לחץ
    const topStressTriggers = Object.entries(this.learningData.stressIndicators.taskTypes)
      .sort(([,a], [,b]) => b - a)
      .slice(0, 3);
```

```
    if (topStressTriggers.length > 0) {
      recommendations.push({
        type: 'stress_management',
        priority: 'high',
        message: `משימות מסוג ${topStressTriggers.map(([type]) => type).join(', ')} | אותן בתקופות רגועות ולהקצות זמן נוסף
        actionable: true,
        action: 'addBufferToStressfulTasks'
      });
    }

    return recommendations;
  }
}
```